THE UNIVERSITY
of EDINBURGH

Connectionist multivariate density-estimation and its application to speech synthesis

Benigno Uria

Doctor of Philosophy

Institute for Adaptive and Neural Computation
School of Informatics
University of Edinburgh

2015

Abstract

Autoregressive models factorize a multivariate joint probability distribution into a product of one-dimensional conditional distributions. The variables are assigned an ordering, and the conditional distribution of each variable modelled using all variables preceding it in that ordering as predictors.

Calculating normalized probabilities and sampling has polynomial computational complexity under autoregressive models. Moreover, binary autoregressive models based on neural networks obtain statistical performances similar to that of some intractable models, like restricted Boltzmann machines, on several datasets.

The use of autoregressive probability density estimators based on neural networks to model real-valued data, while proposed before, has never been properly investigated and reported. In this thesis we extend the formulation of neural autoregressive distribution estimators (NADE) to real-valued data; a model we call the real-valued neural autoregressive density estimator (RNADE). Its statistical performance on several datasets, including visual and auditory data, is reported and compared to that of other models. RNADE obtained higher test likelihoods than other tractable models, while retaining all the attractive computational properties of autoregressive models.

However, autoregressive models are limited by the ordering of the variables inherent to their formulation. Marginalization and imputation tasks can only be solved analytically if the missing variables are at the end of the ordering. We present a new training technique that obtains a set of parameters that can be used for any ordering of the variables. By choosing a model with a convenient ordering of the dimensions at test time, it is possible to solve any marginalization and imputation tasks analytically.

The same training procedure also makes it practical to train NADEs and RNADEs with several hidden layers. The resulting deep and tractable models display higher test likelihoods than the equivalent one-hidden-layer models for all the datasets tested.

Ensembles of NADEs or RNADEs can be created inexpensively by combining models that share their parameters but differ in the ordering of the variables. These ensembles of autoregressive models obtain state-of-the-art statistical performances for several datasets.

Finally, we demonstrate the application of RNADE to speech synthesis, and

confirm that capturing the phone-conditional dependencies of acoustic features improves the quality of synthetic speech. Our model generates synthetic speech that was judged by naive listeners as being of higher quality than that generated by mixture density networks, which are considered a state-of-the-art synthesis technique.

Lay summary

In this thesis we introduce a new set of computational models that can learn the dependencies among all variables in datasets with tens or hundreds of interrelated variables. The models introduced learn these dependencies by repeatedly being presented with observations from the dataset.

Capturing dependencies among variables is an important task in data science that has many commercial and scientific applications. A good model of this kind would be able to generate new data (e.g. for synthesizing speech for digital assistants), denoise measurements (e.g. for denoising digital photographs) or classify new measurements into different classes (e.g. for speech recognition).

The techniques presented in this thesis are able to model the dependencies among variables in datasets of small image patches and speech acoustic recordings. Very importantly, the techniques developed have mild computational cost compared to other models with similar capabilities. As a practical demonstration of the abilities of our models, we applied them to text-to-speech synthesis and obtained higher quality synthetic speech than other state-of-the-art methods.

Acknowledgements

First and foremost, I would like to thank my family and especially my parents. Their encouragement and support in all my decisions has been fundamental in all my academic achievements.

Secondly, I would like to thank my supervisors. If any merit should be ascribed to the author of this thesis, it must be in his good taste at choosing his main supervisor: Iain Murray. Iain kindled my interest in unsupervised modelling and wisely guided all my actions without asphixiating my curiosity. I was also fortunate to have Steve Renals as my second supervisor. Steve's breadth of knowledge proved of great help. My third supervisor, Amos Storkey, provided useful comments during our yearly meetings and words of wisdom during our coffee breaks. Thanks also to the rest of the Adaptive and Neural Computation Institute members for providing a constructive and open research environment.

Thirdly, I would like to thank my financial sponsors: EPSRC and Apple. I was lucky to have John Bridle as my mentor at Apple. John's guidance and wisdom will have a lasting effect in my approach to research. I would also like to thank the rest of Apple's staff in Cheltenham, specially Hywel Richards with whom I had many productive conversations, and Shoichiro Yamanashi with whom I had many beers.

I would also like to thank my friends and colleagues at University of Edinburgh. They made my four year stay there not only bearable, but at times enjoyable. I hope we will have many more brews, espresso martinis, and scotchs together in years to come.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Benigno Uria*)

Table of Contents

Chapter 1

Introduction

Agents, that is, entities that take decisions, seldom act with complete certainty. Uncertainty can have multiple causes: the information available to an agent is usually incomplete or unreliable, and the agent's ability to predict the consequences of its actions is limited. The process by which an agent decides its actions can be divided in two steps (Markowitz, 1952): (1) what to *believe* and, (2) what to *do* given its beliefs. The former is the object of probability theory, the latter of utility theory. In this thesis we restrict ourselves to the study of beliefs. Action is outside the scope of our investigations.

In this thesis we will use probability distributions to represent uncertainty. A probabilistic representation of subjective belief (de Finetti, 1974), and the rules of probability calculus, are the only coherent representation of uncertain belief that satisfy a small set of desirable properties (Jeffreys, 1939; Jaynes, 2003).

In machine learning, *unsupervised learning*, attempts to obtain a joint probability model over all relevant variables. Given such a model, an agent can solve any inference task, i.e. calculate any conditional distribution, using the rules of probability. In other words, the agent can update its belief about a subset of variables based on new information about some other variables.

In practice, it is often the case that a single inference task is to be solved, and this task is known in advance. In these cases, the appropriate conditional distribution can be modelled directly; saving the effort of modelling the relationships among predictor variables. However, the variables predicted may not be conditionally independent and require the use of *structured prediction*. Structured prediction (Taskar, 2004), which is closely related to unsupervised learning, models the probability distribution among the dependant variables of a conditional

distribution.

In this thesis we present a new class of unsupervised models (Chapter 3) for which any inference task can be solved efficiently (Chapter 4). We also show how these models can be easily adapted to the structured prediction regime (Chapter 5).

Judging the quality of the models we introduce is not a simple matter. The choice of a probabilistic model is an empirical science. The usefulness of a model in making future predictions cannot be judged *a priori*, that is, regardless of the data that is modelled. All families of models have inductive biases (Gordon and Desjardins, 1995; Wolpert, 1996), i.e. assumptions that favour one generalization decision over others (Mitchell, 1980). Therefore, every assertion about the statistical performance of a family of models should always be qualified by the data or types of data to which it applies. However, models can be judged *a priori* on their computational performance on different tasks like density estimation, inference, or sampling.

The empirical selection of models is usually done based on their likelihood (Fisher, 1922), or in a Bayesian framework their *a posteriori* probability (Gelman and Shalizi, 2010; MacKay, 2003), for the data modelled. Therefore, in order to perform model selection, it is highly desirable to have probabilistic models for which calculating their likelihood is computationally feasible.

In this thesis we focus our attention on connectionist[1] autoregressive models. That is, models that utilize neural networks to model each of the conditionals of an autoregressive factorization of the joint distribution.

Supervised models based on neural networks are state-of-the-art techniques for the classification of images and speech (Mohamed et al., 2009; Krizhevsky et al., 2012). In this thesis we hypothesize, and corroborate, that the inductive biases of neural networks are also appropriate for the unsupervised modelling of those types of data.

Sampling and density estimation of an entire datapoint are tractable under an autoregressive model, as long as each of the one-dimensional conditional distributions of which it is composed are tractable. This makes it possible to calculate the likelihood of autoregressive models and perform model comparison with other tractable models. However, marginalizing or sampling subsets of

[1] The term "connectionism" is less loaded with biological connotations than the term "artificial neural network". Therefore our preference for this term in the title of this work.

dimensions conditioned on the value of other dimensions is only tractable in the few particular cases where the dimensions predicted are at the end of the autoregressive ordering. In this thesis, we introduce a technique that allows for easy and tractable marginalization and sampling of subsets of dimensions.

A detailed list of contributions follows.

1.1 List of contributions

In Chapter 3 we introduce the *real-valued neural autoregressive density estimator* (RNADE), an extension of the *neural autoregressive distribution estimator* (NADE) (Larochelle and Murray, 2011) , that can model real-valued data. We also analyze the statistical performance of RNADE for natural image patches, speech acoustics, and small datasets of heterogeneous variables.

In Chapter 4 we present a new training procedure, able to obtain a set of parameters for NADE or RNADE models that can be used for any ordering of the variables in the autoregressive factorization of the joint distribution. Using this set of parameters, a model with a convenient ordering of the dimensions can be chosen at test time to solve exactly and tractably any inference or conditional sampling tasks. The same training procedure also makes training deep NADE or RNADEs computationally feasible.

In Chapter 5, a conditional version of RNADE is introduced and used for the structured prediction of speech acoustics conditioned on phonetic labels. This model is able to improve the quality of text-to-speech synthesis compared to unstructured prediction models like mixture density networks.

Chapter 2

Connectionist density estimation

In this chapter we review some of the most popular generative models in the field of machine learning, with an emphasis on connectionist methods. The low statistical peformance of mixture models (Section 2.3) for perceptual data and the low computational performance of Boltzmann machines (Section 2.4) and density-networks (Sections 2.5, 2.6) serve as motivation for the introduction of extensions to autoregressive generative models (Section 2.8) in the following chapters.

2.1 Density estimation

A function f is said to be a *probability density function*[1] of the D-dimensional continuous random variable $\boldsymbol{x} \in \mathbb{R}^D$ if it satisfies:

$$P(\boldsymbol{x} \in A) = \int_A f(\boldsymbol{x}) d^D\boldsymbol{x} \quad \text{for all measurable subsets } A \subset \mathbb{R}^D. \tag{2.1}$$

where $P(\boldsymbol{x} \in A)$ is the probability that an outcome, $\boldsymbol{x}$, will belong to A.

For discrete variables the probability of a particular outcome, a, is well defined and we also have $f(a) = P(\boldsymbol{x} = a)$, in which case the probability density function can also be called probability mass function. We will use the term density function in both cases.

Following Silverman (1986), a *density estimator* is an approximation of a random variable's density function constructed from a dataset of observations (also called training dataset from here on).

Density estimators are sometimes called *generative models* in the machine learning literature. In practice some generative models cannot be used to calculate

[1] A more rigorous definition of a probability density function can be found in Casella and Berger (2002) or from a measure theoretic perspective in Billingsley (2008).

normalized probability densities, as it would require an intractable number of operations. We will use both terms indistinctly, but we will favour the term "density estimator" when the aim is to obtain normalized probability densities.

Joint probability density estimation of multivariate data is one of the most fundamental challenges in machine learning. The vast majority of tasks in machine learning can be posed as doing inference on a generative model of the relevant variables. That is, calculating the probability of an event, A in (2.1). Usually[2], an inference task involves calculating the distribution of a subset of variables, conditioned on a known value of a subset of the rest of variables; while the remaining variables are *marginalized* i.e. integrated out.

Let us call the subset of interest x_I, the subset conditioned on x_C, and the variables to be marginalised x_N (for nuisance). Following, the rules of probability calculus, inference can be posed exclusively in terms of the joint probability density function:

$$p(\boldsymbol{x}_I \,|\, \boldsymbol{x}_C) = \frac{p(\boldsymbol{x}_I, \boldsymbol{x}_C)}{p(\boldsymbol{x}_C)} \tag{2.2}$$

$$= \frac{\int p(\boldsymbol{x}_I, \boldsymbol{x}_C, \boldsymbol{x}_N) d\boldsymbol{x}_N}{\int\int p(\boldsymbol{x}_I, \boldsymbol{x}_C, \boldsymbol{x}_N) d\boldsymbol{x}_I d\boldsymbol{x}_N} \,. \tag{2.3}$$

A non-exhaustive list of tasks that can be addressed by inference on generative models includes:

Classification. Also called discriminant analysis, can be done by fitting a density estimator for each possible class. The posterior distribution over classes for new observations can be obtained using Bayes' rule. This approach to classification can easily deal with missing data by marginalizing over the missing dimensions (e.g. Duda et al., 2001). Optionally, this method allows detecting events with a very low probability density under all classes. The system may reject classifying these *outliers*, for which, for example, human intervention may be preferred.

Sampling. Given a density estimator it is possible to generate new data consistent with the distribution of the training dataset.

Imputation. Sampling any missing variables in an observation, generating plausible complete instances, is possible.

[2]This illustration does not cover all possible types of events. For example, a variable may be known to lie in a certain range which would require integrating only over that range.

Compression. Using arithmetic coding, compression can be reduced to obtaining a good density estimator (e.g. MacKay, 2003, p. 111).

Denoising. Given a good generative model of the data and of the noise process, it is possible to obtain a distribution over the original data (e.g. Zoran and Weiss, 2011).

Given expert knowledge about the process that generates the data to be modelled, a density estimator based on graphical models can be designed (e.g. Koller and Friedman, 2009). Given such a model, standard inference techniques, like belief propagation, sampling, and variational approximations, can be used. However, sometimes the actual generative process, or the mathematical form of parts of it, is unknown, or too complex. In those cases it may be appropriate to use a black-box density estimator i.e. a general-purpose model that ignores the actual process that generated the data.

In this thesis we will only consider black-box density estimators. Nonetheless, these black-box models can also act as modules in generative models based on expert knowledge, or as auxiliary models in sampling or variational techniques (e.g. Bornschein and Bengio, 2014).

In this thesis we will only consider parametric density estimators. That is, models that have a fixed number of parameters, decided by the user before observing the training dataset[3]. In non-parametric models, the number of parameters grows with the size of the dataset used to fit the model. Readers interested in the non-parametric approach to density estimation may consult Silverman (1986).

The main characteristic by which density estimators are usually judged and compared is their statistical performance, that is, their ability to capture the data generating distribution. In this thesis, we will measure the statistical performance of the different models by their marginal log-likelihood (Fisher, 1922). That is, by the average log-probability density they assign to the training data (or a held-out subset of it not used during training). An extension of our efforts to the Bayesian framework would encounter the usual complications, which are more often computational than conceptual (Neal, 1995; MacKay, 1995a).

[3]Note, however, that for models that can use a variable number of parameters it is common to crossvalidate this number. That is, the statistical performance of models with different number of parameters is compared on a held out subset of the training data, and the model with the best performance on this subset is chosen.

In practice, the computational requirements, in time and memory, of a model may be as important as its statistical performance. For a particular model, the computational complexity of sampling, density estimation, and marginalization may be vastly different. As a consequence, different models may be appropriate for different applications. In this thesis we will develop models with attractive computational complexity properties, having low-polynomial complexity for density estimation, sampling and inference.

An exhaustive review of the field of density estimation is beyond the scope of this thesis. The interested reader may consult the many textbooks written on the topic. Both Barber (2012) and Koller and Friedman (2009) are thorough introductions to the use of graphical models for multivariate modelling. Silverman (1986) provides an introduction to non-parametric density estimation. Jaworski et al. (2010) review the copula approach to multivariate modelling. An interesting introduction to the connectionist approach can be found in Frey (1998).

In the rest of this chapter, we will limit ourselves to present some of the most popular density estimators used by the machine learning community; with a special focus on connectionist methods. The aim will not be a detailed description of the intricacies of using and fitting these models, but to display their strong and weak points, particularly, regarding their computational performance. This will motivate our introduction of a new family of density estimators that, we believe, occupies a different region of the statistical performance *vs* computational performance spectrum.

2.2 Gibbs distributions

Energy models assign an energy value, $E(\boldsymbol{x}) \in \mathbb{R}$, to each configuration of a D-dimensional variable of interest $\boldsymbol{x}$. Energy models whose exponentiated negative energy integral over the domain of $\boldsymbol{x}$ is finite can be interpreted as probability distributions whose probability density function is given by the Gibbs distribution:

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp\left\{-E(\boldsymbol{x})\right\}, \quad \text{where} \quad Z = \int \exp\left\{-E(\boldsymbol{x})\right\} d^D\boldsymbol{x}\,. \tag{2.4}$$

The integral is substituted by a sum if the domain of $\boldsymbol{x}$ is discrete. Values of $\boldsymbol{x}$ with higher energy have lower probability. Energy models whose exponentiated integral is not finite cannot be interpreted as probabilistic models, but may still be of use for tasks like classification (LeCun et al., 2007). Any probability model

such that $p(\boldsymbol{x}) > 0$ for all values of $\boldsymbol{x}$ can be posed as an energy model by defining its energy function as $E(\boldsymbol{x}) = -\log p(\boldsymbol{x})$.

For some energy models, normalized density estimation is not computationally tractable. The calculation of Z may require inordinate amounts of computation. The model is only tractable when a closed-form solutions to the integral in (2.4) exists; or in the case of discrete data, when the number of terms in the sum is sufficiently small.

A well known example of a tractable energy model is the multivariate Gaussian which assigns to each point, $\boldsymbol{x} \in \mathbb{R}^D$, an energy equal to a quadratic distance, determined by a $D \times D$ positive definite covariance matrix $\boldsymbol{\Sigma}$, from a point $\boldsymbol{\mu} \in \mathbb{R}^D$, called the mean:

$$E(x) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}) \,. \tag{2.5}$$

Simple generative models like the multivariate Gaussian have attractive computational and statistical characteristics: calculating the density of an observation, sampling, and parameter fitting are all computationally cheap.

The Boltzmann machine (Ackley et al., 1985; Hinton and Sejnowski, 1986) is an energy model over multivariate binary data that, like the Gaussian, has a quadratic energy function. Unfortunately, as we will see in section 2.4, it does not share the computational tractability properties of the Gaussian.

Multivariate Gaussian distributions and Boltzmann machines can only attempt to capture the first and second order moments of the data generating distribution. A more elaborate energy function is required in order to capture further properties of the data generating distribution.

Most of the models we review in this chapter increase their modelling capacity by introducing extra random variables called *hidden* or *latent* variables that are not part of the set of observed variables that we are interested in modelling. The introduction of latent variables comes at the cost of more expensive inference procedures: the configuration of the latent variables has to be marginalized in order to calculate the probabilities of the visible variables. Thus, the increase in modelling capacity is due to the introduction of a log-sum-exp (e.g. Murphy, 2012, p. 86) expression in the energy function.

An alternative to the introduction of stochastic latent variables is the use of an autoregressive model. These models predict the distribution over visible variables one dimension at a time, following a particular order. Each dimension is modelled

using a conditional distribution that uses the value of the dimensions already predicted as explanatory variables. Any joint probability distribution can be posed as an autoregressive model. The modelling capacity of autoregressive models depends on the capacity of the function that maps the values of the preceding dimensions to the parameters of the single dimension conditional distribution modelled at each step. As we will see in Section 2.8.3 and Chapters 3 and 4, the use of flexible functions, like artificial neural networks, whose parameters can be optimized to maximize the likelihood of the resulting model can lead to high statistical performance on many datasets of interest. Moreover, as long as each of these one-dimensional conditionals distributions is computationally tractable, the joint distribution will also be tractable.

2.3 Mixture models

Mixture models increase statistical flexibility by introducing a single discrete-valued latent variable, $c \in \{1 \dots C\}$, that acts as a switch that decides which among C possible models is used to generate the observed variables. A Bayes network depicting a mixture model is shown in Figure 2.1. The marginal distribution over the observations can be calculated by

$$p(\boldsymbol{x}) = \sum_{c=1}^{C} p(c)p(\boldsymbol{x} \,|\, c)\,. \tag{2.6}$$

We note that this factorisation is just an application of the sum and product rules of probability, and any distribution (with discrete hidden variable c) satisfies it. However, mixture models are usually explicitly parameterized in terms of this factorisation[4]. With a set of parameters, $\boldsymbol{\pi}$, for $p(c)$ and a different set of parameters for each conditional distribution of the observations given the hidden variable $p(\boldsymbol{x} \,|\, \boldsymbol{\theta}_c)$. Each of the conditional distributions is called a *component*, and the probability mass for each value of c is called the *component weight*, where the component weights must be positive and sum to 1.

When used as a black-box density estimator, it is common to use the same parametric form for all the components in the mixture. However, components belonging to different families can be used if domain knowledge indicates this might be a good idea.

[4] In Section 2.4 we will present the restricted Boltzmann machine: a model that can be interpreted as a mixture model, but whose parameterization does not explicitly follow the factorisation in (2.6).

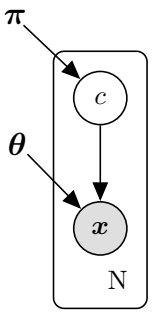

Figure 2.1: Bayes network graphical model of a mixture model. For each datapoint, the discrete variable c identifies the component responsible for generating each of the N observations $\boldsymbol{x}$.

Mixture models are computationally inexpensive. Calculating the probability density of an observation under a mixture model is tractable (as long as it is so under each of its components, growing linearly with the number of components). In consequence, model likelihoods can also be computed tractably; allowing the comparison of their statistical performance to that of other models.

Unfortunately, mixture models can have low statistical performance for a common class of datasets. When modelling data that originates from a compositional generative model. That is, if the data is created by the non-exclusive activation and deactivation of characteristics. In that case, an exponential number of components may be necessary, one for each possible combination of the characteristics. This type of dataset is, generally, better modelled by autoregressive models (see Section 2.8) or products of experts like the restricted Boltzmann machine (see Section 2.4).

The parameters of mixture models are usually fitted to maximize the marginal likelihood of the model for the observed data $\sum_c p(\boldsymbol{x}, c)$. Even for simple parametric forms of the components, this maximization problem cannot be solved analytically[5]. Gradient descent methods can be used to fit the parameters, but usually the *expectation-maximization* (EM) algorithm (Dempster et al., 1977) is used.

In the next section we will review the EM-algorithm in its general variational form. Following which, we will review some of the most commonly used mixture models, namely: the mixture of multivariate Bernoulli distributions, the mixture of Gaussians, and the mixture of factor analysers. These models play an important role in the subsequent chapters as they will serve as baseline tractable models for

[5]For certain restrictive classes of mixtures, the parameters of the model can be approximately recovered from samples using spectral methods. See, for example, Hsu and Kakade (2013).

our experiments.

2.3.1 The EM-algorithm

The expectation-maximization (or simply EM) algorithm (Dempster et al., 1977) is an iterative parameter fitting procedure used in the presence of missing or truncated data. The EM-algorithm is therefore of use in training mixture models, where the identity of the component used to generate each observation is unknown. Each iteration of the EM algorithm is guaranteed not to decrease a lower bound on the likelihood of the model, and therefore will converge to a parameter configuration that is a, possibly local, maximum of the likelihood landscape.

Following Barber (2012), let $p(\boldsymbol{x}, c \,|\, \boldsymbol{\theta})$ be the distribution whose parameters, $\boldsymbol{\theta}$ we would like to fit in order to maximize their marginal likelihood with respect to some training data $D = \left\{\boldsymbol{x}^{(n)}\right\}_{n=1}^{N}$. To keep the notation tidy we will show the expressions for a single datapoint $\boldsymbol{x}$. Let us start by introducing a conditional distribution $q(c \,|\, \boldsymbol{x})$ and calculating its KL-divergence with the conditional distribution $p(c \,|\, \boldsymbol{x}, \boldsymbol{\theta})$:

$$\begin{aligned} \mathrm{KL}\left(q(c \,|\, \boldsymbol{x}) \parallel p(c \,|\, \boldsymbol{x}, \boldsymbol{\theta})\right) &= \left\langle \log q(c \,|\, \boldsymbol{x}) \right\rangle_q - \left\langle \log p(c \,|\, \boldsymbol{x}, \boldsymbol{\theta}) \right\rangle_q && (2.7) \\ &= \left\langle \log q(c \,|\, \boldsymbol{x}) \right\rangle_q - \left\langle \log p(\boldsymbol{x}, c \,|\, \boldsymbol{\theta}) \right\rangle_q + \log p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) \,. && (2.8) \end{aligned}$$

Rearranging the terms we can obtain an expression for the marginal likelihood of the parameters, $\boldsymbol{\theta}$, for an observation $\boldsymbol{x}$:

$$\log p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \left\langle \log p(\boldsymbol{x}, c \,|\, \boldsymbol{\theta}) \right\rangle_q - \left\langle \log q(c \,|\, x) \right\rangle_q + \mathrm{KL}\left(q(c \,|\, \boldsymbol{x}) \parallel p(c \,|\, \boldsymbol{x}, \boldsymbol{\theta})\right) . \tag{2.9}$$

Kullback-Leibler divergences are non-negative(e.g. MacKay, 2003), therefore we can obtain a lower bound on the likelihood of the parameters $\boldsymbol{\theta}$:

$$\log p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) \geq \left\langle \log p(\boldsymbol{x}, c \,|\, \boldsymbol{\theta}) \right\rangle_q - \left\langle \log q(c \,|\, x) \right\rangle_q . \tag{2.10}$$

Backed by this result, the EM-algorithm proceeds by iteratively repeating the following two steps to convergence:

E-step With fixed $\boldsymbol{\theta}^{(t-1)}$, optimize $q^{(t)}(c \,|\, \boldsymbol{x})$ to maximize the lower-bound (2.10).

M-step With fixed $q^{(t)}(c \,|\, \boldsymbol{x})$, optimize $\boldsymbol{\theta}^{(t)}$ to maximize the lower-bound (2.10).

for an arbitrary initialization of the parameters of p, $\boldsymbol{\theta}^{(0)}$.

The EM algorithm, in its general variational form presented thus far, is guaranteed not to decrease the likelihood lower bound with each step, but offers no guarantees on the actual likelihood of the parameters. However, if $p(c\,|\,\boldsymbol{x},\boldsymbol{\theta}^{(t-1)})$ is tractable, then it can be used as the variational distribution $q^{(t)}$ at each E-step. In this case, the EM algorithm is guaranteed to obtain a sequence of parameters $\boldsymbol{\theta}^{(1)},\boldsymbol{\theta}^{(2)},\ldots,\boldsymbol{\theta}^{(t)}$ with non-decreasing likelihoods for $\boldsymbol{x}$. We call this version of the algorithm classic-EM, or simply EM. To prove that these updates will not decrease the marginal likelihood, we can calculate the difference in marginal likelihood between two consecutive configurations of $\boldsymbol{\theta}$; using (2.9):

$$\begin{aligned}\log p(\boldsymbol{x}\,|\,\boldsymbol{\theta}^{(t)}) =& \left\langle \log p(\boldsymbol{x},c\,|\,\boldsymbol{\theta}^{(t)}) \right\rangle_{p_{\boldsymbol{\theta}^{(t)}}} - \left\langle \log p(c\,|\,\boldsymbol{x},\boldsymbol{\theta}^{(t)}) \right\rangle_{p_{\boldsymbol{\theta}^{(t)}}} \\ &+ \mathrm{KL}\left(p(c\,|\,x,\boldsymbol{\theta}^{(t)}) \parallel p(c\,|\,x,\boldsymbol{\theta}^{(t)})\right), \qquad (2.11)\\ \log p(\boldsymbol{x}\,|\,\boldsymbol{\theta}^{(t+1)}) =& \left\langle \log p(\boldsymbol{x},c\,|\,\boldsymbol{\theta}^{(t+1)}) \right\rangle_{p_{\boldsymbol{\theta}^{(t)}}} - \left\langle \log p(c\,|\,\boldsymbol{x},\boldsymbol{\theta}^{(t)}) \right\rangle_{p_{\boldsymbol{\theta}^{(t)}}} \\ &+ \mathrm{KL}\left(p(c\,|\,x,\boldsymbol{\theta}^{(t)}) \parallel p(c\,|\,x,\boldsymbol{\theta}^{(t+1)})\right). \qquad (2.12)\end{aligned}$$

The KL-divergence in (2.11) is zero, and the second term in each r.h.s cancel out. Therefore,

$$\begin{aligned}\log p(\boldsymbol{x}\,|\,\boldsymbol{\theta}^{(t+1)}) - \log p(\boldsymbol{x}\,|\,\boldsymbol{\theta}^{(t)}) =& \underbrace{\mathrm{KL}\left(p(c\,|\,x,\boldsymbol{\theta}^{(t)}) \parallel p(c\,|\,x,\boldsymbol{\theta}^{(t+1)})\right)}_{\geq 0} \\ &+ \underbrace{\left\langle \log p(\boldsymbol{x},c\,|\,\boldsymbol{\theta}^{(t+1)}) \right\rangle_{p_{\boldsymbol{\theta}^{(t)}}} - \left\langle \log p(\boldsymbol{x},c\,|\,\boldsymbol{\theta}^{(t)}) \right\rangle_{p_{\boldsymbol{\theta}^{(t)}}}}_{\geq 0}. \qquad (2.13)\end{aligned}$$

The KL-divergence term is non-negative, and given that $\boldsymbol{\theta}^{(t+1)}$ has been optimized in the M-step to maximize the joint probability of the observations under the distribution $p(c\,|\,\boldsymbol{x},\boldsymbol{\theta}^{(t)})$, the difference between the third and second terms in (2.13) must also be non-negative. Therefore, the difference in likelihoods must be non-negative.

The classic EM algorithm guarantees convergence of the parameters to a local maximum of the likelihood function. Therefore, as with most local optimization methods, it may benefit from multiple runs with different $\boldsymbol{\theta}^{(0)}$ initializations.

It is possible to use the classic EM algorithm to train a mixture model. Under a mixture model, $p(c\,|\,\boldsymbol{x},\boldsymbol{\theta}^{(t)})$ can be computed (with a computational complexity linear in the number of components) by calculating the probability density of

the observations under each component and renormalizing to obtain a valid distribution over components:

$$p(c\,|\,\boldsymbol{x},\boldsymbol{\theta}^{(t)}) = \frac{p(\boldsymbol{x}\,|\,c,\boldsymbol{\theta}^{(t)})p(c\,|\,\boldsymbol{\theta}^{(t)})}{\sum_{c'} p(\boldsymbol{x}\,|\,c',\boldsymbol{\theta}^{(t)})p(c'\,|\,\boldsymbol{\theta}^{(t)})}\,. \tag{2.14}$$

To improve the computational performance of EM when the training dataset is very big, a stochastic minibatch version is sometimes used (Zoran and Weiss, 2011). The intuition behind this technique is that calculating the EM updates for a randomly chosen subset, $\mathcal{B}$, of the training dataset may be informative enough to move to a better configuration of the parameters; especially in the first few iterations when the parameters have been initialized randomly. In order not to overfit to the subset of training datapoints chosen for each update, the parameters are not substituted by the parameters obtained in the M-step for the minibatch $\mathcal{B}$. Instead, a convex linear combination of the previous configuration and the new parameters is used:

$$\hat{\boldsymbol{\theta}}_{\mathcal{B}} \leftarrow EM(\boldsymbol{\theta}^{(t-1)},\mathcal{B})\,, \tag{2.15}$$

$$\boldsymbol{\theta}^{(t)} = (1-\eta_t)\boldsymbol{\theta}^{(t-1)} + \eta_t\hat{\boldsymbol{\theta}}_{\mathcal{B}}\,. \tag{2.16}$$

Where the size of the step, η_t, used at each iteration is annealed in order to guarantee convergence.

2.3.2 Mixtures of multivariate Bernoullis

Multivariate binary data, $\boldsymbol{x} \in \{0,1\}^D$, can be modelled using a mixture of multivariate Bernoulli distributions. Each multivariate Bernoulli distribution assumes a factorial distribution over the D visible dimensions, and is defined as:

$$mvBern(\boldsymbol{x}\,|\,\boldsymbol{\mu}) = \prod_{d=1}^{D} \mu_d^{x_d}(1-\mu_d)^{(1-x_d)}\,. \tag{2.17}$$

That is, the probability of each dimension taking value 1 is independent of the value of the rest of dimensions.

Therefore, the probability of a particular observation $\boldsymbol{x}$ under a mixture of C multivariate Bernoulli distributions, each component having parameters $\boldsymbol{\mu}_c \in (0,1)^D$ and weight $\pi_c \in [0,1]$, is given by:

$$p(\boldsymbol{x}\,|\,\boldsymbol{\pi},\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_C) = \sum_{c=1}^{C} \pi_c\; mvBern(\boldsymbol{x}\,|\,\boldsymbol{\mu}_c)\,. \tag{2.18}$$

Training is usually done using the EM algorithm, with updates:

E-step :

$$p(c\,|\,\boldsymbol{x}^{(n)},\boldsymbol{\theta}^{(t-1)}) \propto \pi_c^{(t-1)} mvBern(\boldsymbol{x}^{(n)}\,|\,\mu_c^{(t-1)})\,. \tag{2.19}$$

M-step :

$$\pi_c^{(t)} = \frac{1}{N}\sum_n p(c\,|\,\boldsymbol{x}^{(n)},\boldsymbol{\theta}^{(t-1)})\,, \tag{2.20}$$

$$\boldsymbol{\mu_c}^{(t)} = \frac{1}{N}\sum_n p(c\,|\,\boldsymbol{x}^{(n)},\boldsymbol{\theta}^{(t-1)})\boldsymbol{x}^{(n)}\,. \tag{2.21}$$

2.3.3 Mixtures of Gaussians

A common mixture model for real-valued data, $\boldsymbol{x} \in \mathbb{R}^D$, is the mixture of Gaussians (MoG). In a mixture of Gaussians, each component follows a multivariate Gaussian distribution:

$$\mathcal{N}(\boldsymbol{x}\,|\,\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\}. \tag{2.22}$$

The probability density of an observation under a mixture of C Gaussians, with component weights π_c, each component having mean $\boldsymbol{\mu}_c \in \mathbb{R}^D$, covariance matrix $\boldsymbol{\Sigma}_c \in \mathbb{R}^{D\times D}$, which must be symmetric and positive definite, is given by:

$$p(\boldsymbol{x}\,|\,\boldsymbol{\theta}) = \sum_{c=1}^{C} \pi_c \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}_c|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_c})^\top \boldsymbol{\Sigma}_c^{-1}(\boldsymbol{x}-\boldsymbol{\mu_c})\right\}. \tag{2.23}$$

Training can be done by gradient descent on the negative log-likelihood of the parameters, but is commonly done using the EM-algorithm, using the steps:

E-step :

$$p(c\,|\,\boldsymbol{x}^{(n)},\boldsymbol{\theta}^{(t-1)}) \propto \pi_c^{(t-1)} \mathcal{N}(\boldsymbol{x}^{(n)}\,|\,\boldsymbol{\mu}_c^{(t-1)},\boldsymbol{\Sigma}_c^{(t-1)})\,. \tag{2.24}$$

M-step :

$$\pi_c^{(t)} = \frac{1}{N}\sum_n p(c\,|\,\boldsymbol{x}^{(n)},\boldsymbol{\theta}^{(t-1)})\,, \tag{2.25}$$

$$\boldsymbol{\mu_c}^{(t)} = \frac{1}{N}\sum_n p(c\,|\,\boldsymbol{x}^{(n)},\boldsymbol{\theta}^{(t-1)})\boldsymbol{x}^{(n)}\,, \tag{2.26}$$

$$\boldsymbol{\Sigma}_c^{(t)} = \frac{1}{N}\sum_n p(c\,|\,\boldsymbol{x}^{(n)},\boldsymbol{\theta}^{(t-1)})(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_c^{(t)})(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_c^{(t)})^\top\,. \tag{2.27}$$

The number of parameters in an unconstrained covariance mixture of Gaussians grows quadratically with the dimensionality of the data. Sometimes the covariance matrix of the components is constrained to be diagonal (axis-aligned-Gaussian components) or even diagonal and shared by all dimensions (isotropic-variance components). Another common constrained covariance form is the factor analyser, presented in the next section.

2.3.4 Mixtures of factor analysers

A mixture of factor analysers (MFA) (Ghahramani and Hinton, 1996) is a mixture of Gaussians where the covariance of each component is constrained to have the form low-rank plus diagonal. The main advantage of a mixture of factor analysers with respect to a mixture of unconstrained Gaussians is its ability to use more components without overfitting, due to the reduced amount of parameters of each component.

Let us start by introducing the factor analyser. Factor analysers (FA) are continuous-latent-variable models used to model real-valued observations. More specifically, they are Gaussian-linear models. A visible variable $\boldsymbol{x} \in \mathbb{R}^D$ is modelled as a linear transformation, parameterized by a matrix $\boldsymbol{F} \in \mathbb{R}^{D \times H}$, of a continuous hidden variable $\boldsymbol{h} \in \mathbb{R}^H$, the factors, plus independent Gaussian noise, $\boldsymbol{\epsilon}$.

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{F}\boldsymbol{h} + \boldsymbol{\epsilon}\,. \tag{2.28}$$

The $D \times H$ matrix $\boldsymbol{F}$ is called the factor loading matrix, and the D-dimensional constant $\boldsymbol{\mu}$ sets the mean of the Gaussian. Without loss of generality, the factors are assumed to follow the unit spherical Gaussian distribution:

$$p(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}\,|\,\boldsymbol{0}, \boldsymbol{I})\,, \tag{2.29}$$

while the Gaussian noise is modelled by a diagonal covariance Gaussian:

$$p(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{\epsilon}\,|\,\boldsymbol{0}, \boldsymbol{\Psi})\,. \tag{2.30}$$

Under a factor analyser, data is expected to lie close to an H-dimensional affine subspace of the visible space. Given that $\boldsymbol{x}$ is linearly related to $\boldsymbol{h}$, and both $\boldsymbol{h}$ and $\boldsymbol{\epsilon}$ are Gaussian distributed, $\boldsymbol{x}$ is Gaussian distributed too:

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}\,|\,\boldsymbol{\mu}, \boldsymbol{F}\boldsymbol{F}^{\mathrm{T}} + \boldsymbol{\Psi})\,. \tag{2.31}$$

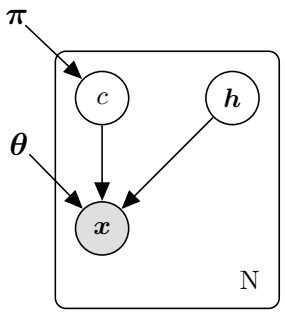

Figure 2.2: Bayes network graphical model of a mixture of factor analysers. For each datapoint, the discrete variable c identifies the component, and $\boldsymbol{h}$ the factors, responsible for generating the observations $\boldsymbol{x}$.

In order to maximize the marginal likelihood of the parameters of a factor analyser, the EM algorithm is used (Rubin and Thayer, 1982) due to the presence of the unobserved factors. The mean parameter is set to the empirical mean, $\boldsymbol{\mu} = \frac{1}{N}\sum_n \boldsymbol{x}^{(n)}$, then the EM-algorithm repeats the following updates to convergence:

E-step :

$$p(\boldsymbol{h}\,|\,\boldsymbol{x}) = \mathcal{N}(\boldsymbol{h}\,|\,\boldsymbol{m}, \boldsymbol{\Sigma}), \tag{2.32}$$

$$\boldsymbol{\Sigma} = (\boldsymbol{I} + \boldsymbol{F}^\top \boldsymbol{\Psi}^{-1} \boldsymbol{F})^{-1}, \tag{2.33}$$

$$\boldsymbol{m} = \boldsymbol{\Sigma} \boldsymbol{F}^\top \boldsymbol{\Psi}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}). \tag{2.34}$$

where we have hidden the superscripts in $\boldsymbol{F}^{(t-1)}$ and $\boldsymbol{\Psi}^{(t-1)}$ to avoid clutter.

M-step :

$$\boldsymbol{F}^{(t)} = \boldsymbol{A}\boldsymbol{H}^{-1}, \tag{2.35}$$

$$\boldsymbol{\Psi}^{(t)} = \mathrm{diag}\left(\frac{1}{N}\sum_n (\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top - 2\boldsymbol{F}\boldsymbol{A}^\top + \boldsymbol{F}\boldsymbol{H}\boldsymbol{F}^\top\right), \tag{2.36}$$

$$\boldsymbol{A} = \frac{1}{N}\sum_n (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}) \langle \boldsymbol{h} \rangle^\top_{p(\boldsymbol{h}\,|\,\boldsymbol{x}^{(n)})}, \tag{2.37}$$

$$\boldsymbol{H} = \frac{1}{N}\sum_n \left\langle \boldsymbol{h}\boldsymbol{h}^\top \right\rangle_{p(\boldsymbol{h}\,|\,\boldsymbol{x}^{(n)})}. \tag{2.38}$$

A mixture of factor analysers has two sets of latent variables: a discrete hidden variable, $c \in 1, \ldots, C$, indicating the component used in generating the data; and the factors, $\boldsymbol{h} \in \mathbb{R}^H$. A Bayes network graphical model of a mixture of factors analysers can be seen in Figure 2.2.

Therefore, the probability density of an observation under an MFA is given by:

$$p(\boldsymbol{x}) = \sum_{c=1}^{C} \pi_c \mathcal{N}(\boldsymbol{x} \,|\, \boldsymbol{\mu}_c, \boldsymbol{F}_c \boldsymbol{F}_c^{\mathrm{T}} + \boldsymbol{\Psi}), \tag{2.39}$$

where π_c are the component weights satisfying $\pi_c > 0$ and $\sum_c \pi_c = 1$, $\boldsymbol{F}_c \in \mathbb{R}^{DxH}$ are the factor loadings of the c-th component, and $\boldsymbol{\Psi}_c = \mathrm{diag}\,(\psi_{c1}, \ldots, \psi_{cD})$ the covariance of the observation noise.

Mixtures of factor analysers can be trained to maximize the marginal likelihood of the model for a training dataset using the EM-algorithm; the parameter update equations can be found in the appendices of Ghahramani and Hinton (1996).

2.4 Boltzmann machines

Boltzmann machines (Ackley et al., 1985) are probabilistic energy models over binary variables, that assign energy $E(x) = -\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{W}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}$ to each configuration $\boldsymbol{x} \in \{0,1\}^D$, where $\boldsymbol{W}$ is constrained to be symmetric and have zero value in all its main diagonal entries.

Boltzmann machines like the one described, where all variables are observed, only capture the first and second order statistics of the data generating distribution. Latent variables are often introduced in order to capture higher order statistics (Hinton and Sejnowski, 1986).

In order to facilitate inference and parameter learning, some off-diagonal entries of the weight matrix $\boldsymbol{W}$ may also be constrained to take value zero. This can be seen as a restriction of connectivity in the Markov net graphical representation of the Boltzmann machine.

2.4.1 Restricted Boltzmann machines

Restricted Boltzmann machines (RBMs) are a type of Boltzmann machine with latent variables and specific type of restricted connectivity. In an RBM variables are partitioned in two sets, called layers. One of the layers includes all visible units, $\boldsymbol{x}$, and the other all hidden units, $\boldsymbol{h}$ (see Figure 2.3). Connectivity between units in the same layer is not permitted. This restricted connectivity allows us to substitute the constrained $(D+H)\times(D+H)$ matrix of the fully connected Boltzmann machine with an unconstrained $D \times H$ matrix, resulting in the energy

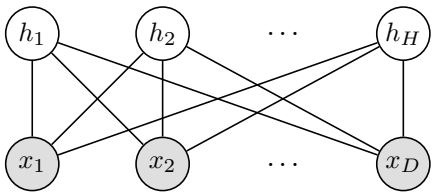

Figure 2.3: Markov network graphical model of a restricted Boltzmann machine. Each unit (variable) is connected to every unit on the other layer and to no units on its own layer.

function:

$$E(\boldsymbol{x}, \boldsymbol{h}) = -\boldsymbol{x}^\top \boldsymbol{W} \boldsymbol{h} - \boldsymbol{b}^\top \boldsymbol{x} - \boldsymbol{c}^\top \boldsymbol{h}, \tag{2.40}$$

where $\boldsymbol{W} \in \mathbb{R}^{D \times H}$, $\boldsymbol{b} \in \mathbb{R}^D$, and $\boldsymbol{c} \in \mathbb{R}^H$ are parameters of the model. The probability of a full configuration of visible and latent units is given by:

$$p(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{h})\right\}, \quad \text{where} \quad Z = \sum_{\boldsymbol{x}, \boldsymbol{h}} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{h})\right\}. \tag{2.41}$$

The probability of a configuration of the visible variables is calculated by marginalizing out the hidden variables:

$$p(\boldsymbol{x}) = \frac{1}{Z} \sum_{\boldsymbol{h}} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{h})\right\} = \frac{1}{Z} \exp\left\{-F(\boldsymbol{x})\right\}, \tag{2.42}$$

where $F(\boldsymbol{x})$ or free energy of $\boldsymbol{x}$ is defined as: $F(\boldsymbol{x}) \equiv -\log \sum_{\boldsymbol{h}} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{h})\right\}$.

Due to the RBM's restricted connectivity, the dimensions in $\boldsymbol{h}$ become independent when conditioned on $\boldsymbol{x}$. As a consequence, the free energy can be efficiently computed in $O(HD)$ time:

$$F(\boldsymbol{x}) = -\boldsymbol{b}^\top \boldsymbol{x} - \sum_{i=1}^{H} \log\left(1 + \exp\left\{\boldsymbol{x}^\top \boldsymbol{W}_{\cdot,i} + c_i\right\}\right). \tag{2.43}$$

However, computing a normalized probability, requires the partition function, Z. The partition function can be calculated by iterating over all possible values of the visible variables; using $Z = \sum_{\boldsymbol{x}} \exp\left\{-F(\boldsymbol{x})\right\}$, which allows us to calculate it with complexity $O(2^D HD)$, which may be feasible if the number of visible variables is sufficiently small ($D \lesssim 40$). By symmetry on the visible and latent roles, if the number or hidden units is small, then the partition function can be calculated in time $O(2^H HD)$.

Therefore, RBMs may be practical for tasks requiring normalized probability estimation, but only if the data is low-dimensional ($D \lesssim 40$). When the data has

a greater number or dimensions, exact calculation of Z will only be feasible if the number of hidden units is small ($H \lesssim 40$). Given that adding hidden units increases the modelling power of RBMs, using a small number of them may be appropriate or not depending on the complexity of the data generating distribution that we are trying to model. As an example, Salakhutdinov and Murray (2008) found that using an RBM with a small number of latent variables to model binary images of handwritten digits has low statistical performance, and is not competitive with other tractable models (Larochelle and Murray, 2011).

When Z is intractable, its value can be approximated using Markov chain Monte Carlo techniques like annealed importance sampling (Salakhutdinov and Murray, 2008; Murray and Salakhutdinov, 2009) or moment-averaging annealed importance sampling (Grosse et al., 2013). However, these techniques tend to underestimate Z which leads to the dangerous conclusion that the model's likelihood is higher than its actual value. Therefore, expertise in the use and tuning of MCMC methods is required when estimating the partition function.

Sampling from an RBM is usually done using block Gibbs sampling (Geman and Geman, 1984) due to the simple expression of the conditionals

$$p(\boldsymbol{x}\,|\,\boldsymbol{h}) = \prod_{i=1}^{D} p(x_i\,|\,\boldsymbol{h}) \;\text{ where }\; p(x_i = 1\,|\,\boldsymbol{h}) = \operatorname{sigm}\left(\boldsymbol{W}_{i,\cdot}\boldsymbol{h} + b_i\right), \tag{2.44}$$

$$p(\boldsymbol{h}\,|\,\boldsymbol{x}) = \prod_{j=1}^{H} p(h_j\,|\,\boldsymbol{x}) \;\text{ where }\; p(h_j = 1\,|\,\boldsymbol{x}) = \operatorname{sigm}\left(\boldsymbol{x}^\top \boldsymbol{W}_{\cdot,j} + c_j\right), \tag{2.45}$$

where $\operatorname{sigm}(\cdot)$ is the logistic sigmoid function defined as:

$$\operatorname{sigm}(x) \equiv \frac{1}{1 + \exp\{-x\}}. \tag{2.46}$$

Although true samples from an RBM can only be obtained asymptotically, samples from the Gibbs chain separated by a few Gibbs steps are commonly used (Hinton, 2010). This requires fitting a parameter that specifies how many Gibbs steps to take between samples, trading computational time for higher independence between successive samples.

The parameters of an RBM are usually fitted to maximize the likelihood of the model given the dataset. Training is usually done by stochastic gradient descent

on the negative marginal log-likelihood of the parameters for the training dataset:

$$-\frac{\partial \log p(\boldsymbol{x})}{\partial \theta_i} = \frac{\partial F(\boldsymbol{x})}{\partial \theta_i} + \frac{\partial \log Z}{\partial \theta_i} \tag{2.47}$$

$$= \frac{\partial F(\boldsymbol{x})}{\partial \theta_i} + \frac{1}{Z} \sum_{\tilde{\boldsymbol{x}}} \frac{\partial \exp\{-F(\tilde{\boldsymbol{x}})\}}{\partial \theta_i} \tag{2.48}$$

$$= \frac{\partial F(\boldsymbol{x})}{\partial \theta_i} + \sum_{\tilde{\boldsymbol{x}}} \frac{1}{Z} \exp\{-F(\tilde{\boldsymbol{x}})\} \frac{-\partial F(\tilde{\boldsymbol{x}})}{\partial \theta_i} \tag{2.49}$$

$$= \frac{\partial F(\boldsymbol{x})}{\partial \theta_i} - \sum_{\tilde{\boldsymbol{x}}} p(\tilde{\boldsymbol{x}}) \frac{\partial F(\tilde{\boldsymbol{x}})}{\partial \theta_i}, \tag{2.50}$$

where $\boldsymbol{x}$ is an observation sampled at random from the training dataset, and the sum over $\tilde{\boldsymbol{x}}$ is done over the whole domain of the visible variables. The sum in (2.50) has an exponential number of terms, and is usually approximated by sampling $\tilde{\boldsymbol{x}}$. However, sampling from the distribution for each gradient update can be too computationally expensive. In practice, a single sample of $\tilde{\boldsymbol{x}}$ obtained by running a few block Gibbs steps initialized from the datapoint $\boldsymbol{x}$ is used. This approximate technique is called contrastive divergence (CD) (Hinton, 2002). In order to obtain a good distribution estimator a considerable number of Gibbs steps (≈ 25) is required (Salakhutdinov and Murray, 2008). An alternative technique, called persistent contrastive divergence (PCD) (Tieleman, 2008), does not reinitialize the Gibbs chain with the data but with the samples of $\tilde{\boldsymbol{x}}$ used in the previous stochastic gradient calculation.

RBMs are powerful models able to capture complicated relationships in binary data (Larochelle and Murray, 2011) (also see comparison tables in Chapter 4). However, due to the intractability of their partition function, they are seldom used for probability estimation. Moreover, training cannot be readily done by gradient descent on the log-likelihood and MCMC techniques are required for sampling. The intractability of RBMs makes it difficult to even crossvalidate how many hidden units are required or what training hyperparameter values provide the best statistical performance.

The most common use for RBMs has been in the pretraining of deep artificial neural networks (DNNs) (see Section 2.4.3), but this procedure now seems unnecessary with the introduction of rectified linear units (Nair and Hinton, 2010; Dahl et al., 2013), better data-independent initialization techniques (Sutskever, 2013) and training algorithms (Martens, 2010).

2.4.2 Gaussian-Bernoulli restricted Boltzmann Machines

Gaussian restricted Boltzmann machines (GRBM) extend restricted Boltzmann machines to model real-valued data (Welling et al., 2005). The observable variables in a GRBM take real values, $\boldsymbol{x} \in \mathbb{R}^D$. The hidden variables are binary valued, $\boldsymbol{h} \in \{0,1\}^H$.

The energy function for a GRBM has the expression:

$$E(\boldsymbol{x}, \boldsymbol{h}) = \sum_{i=1}^{D} \frac{(x_i - b_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H} c_j h_j - \sum_{i=1}^{D} \sum_{j=1}^{H} \frac{x_i}{\sigma_i} W_{i,j} h_j \,, \tag{2.51}$$

where $\boldsymbol{W} \in \mathbb{R}^{D \times H}$, $\boldsymbol{b} \in \mathbb{R}^D$, $\boldsymbol{c} \in \mathbb{R}^H$, $\boldsymbol{\sigma} \in \mathbb{R}^{+D}$ are parameters of the model. The conditional distribution of each layer conditioned on the other are given by:

$$p(x_i \,|\, \boldsymbol{h}) = \mathcal{N}\left(x_i \,|\, b_i + \sigma_i \sum_{j=1}^{H} h_j W_{i,j}, \sigma_i^2\right) , \tag{2.52}$$

$$p(h_j = 1 \,|\, \boldsymbol{x}) = \text{sigm}\left(c_j + \sum_{i=1}^{D} \frac{x_i}{\sigma_i} W_{i,j}\right) . \tag{2.53}$$

The visible units follow independent Gaussian distributions conditioned on the value of the hidden variables; while the latent variables factorize into independent Bernoulli distributions conditioned on the value of the visible units.

The parameters of a Gaussian-RBM are fitted using the contrastive divergence approximation to the log-likelihood gradient. In practice, learning the value of the standard deviation parameter of each variable is difficult. The learning rate for the σ_i parameters needs to be 3 or 4 orders of magnitude smaller than for the rest of parameters (Krizhevsky and Hinton, 2009). For this reason, all σ_i are often fixed to unity (Cho et al., 2011). Note, that this is usually done for data that has already been standardized to have variance 1, which results in models that overestimate the conditional standard deviation and whose samples look noisy.

Cho et al. (2011) reformulated the Gaussian RBM to facilitate learning of the conditional standard deviations, however it still requires the use of complex parallel tempering training procedures.

The mean-covariance RBM (Ranzato and Hinton, 2010) is a generalization of the GRBM that allows for non-diagonal covariance structure for the $p(\boldsymbol{x} \,|\, \boldsymbol{h})$ conditionals. The spike-and-slab RBM (Courville et al., 2011) introduces a set of continuous-valued latent variables associated to each binary latent variable. However, inference using these models is even computationally costlier.

In summary, Gaussian-RBMs suffer from the same drawbacks as binary RBMs, namely: calculating probability densities is intractable (thus exact likelihoods cannot be calculated), and sampling requires MCMC methods. Moreover, Gaussian-RBMs are limited by the diagonal covariance distribution of the visible units conditioned on the hidden units and the difficulty to fit its variance parameters.

2.4.3 Deep belief networks

Due to the discrete domain of the hidden variables, the distribution modelled by an RBM can be interpreted as a mixture model with an exponential number of components; with each configuration of the hidden variables corresponding to a component of the mixture

$$p(\boldsymbol{x}\,|\,\boldsymbol{\theta}) = \sum_{\boldsymbol{h}} p(\boldsymbol{h}\,|\,\boldsymbol{\theta}) p(\boldsymbol{x}\,|\,\boldsymbol{h}, \boldsymbol{\theta})\,, \tag{2.54}$$

where $p(\boldsymbol{h})$ are the component weights, and $p(\boldsymbol{x}\,|\,\boldsymbol{h})$ the distribution of each component. Note that here, we have made the parameters of the RBM, $\boldsymbol{\theta}$, explicit.

In the case of binary RBMs, each component is a multivariate Bernoulli distribution where the probability of each visible variable is given by (2.44). In the case of Gaussian-Bernoulli RBMs, each component is a Gaussian whose parameter values are given by (2.52).

However, an RBM cannot represent any mixture model. The parameters of all components are tied, i.e. calculated using the reduced number of parameters: $\boldsymbol{\theta}$. For both the binary and the Gaussian-RBM, the means of the components are not free to take any set of values. They always correspond to a projection of the vertices of a hypercube[6] of dimensionality H onto the space of observables of dimensionality D.

The component weights are also calculated using the same RBM parameters used to calculate the component parameters, and can be obtained by marginalizing the visible variables (Wang et al., 2012).

Deep belief networks (DBN), as presented in Hinton et al. (2006), aim to increase the statistical performance of RBMs by untying the parameters used to calculate the component weights from those used to calculate the parameters of each component. That is, DBNs introduce a new distribution over $\boldsymbol{h}$ with a

[6] In the binary RBM it is the logit of the means that lies on the projection of a hypercube's vertices.

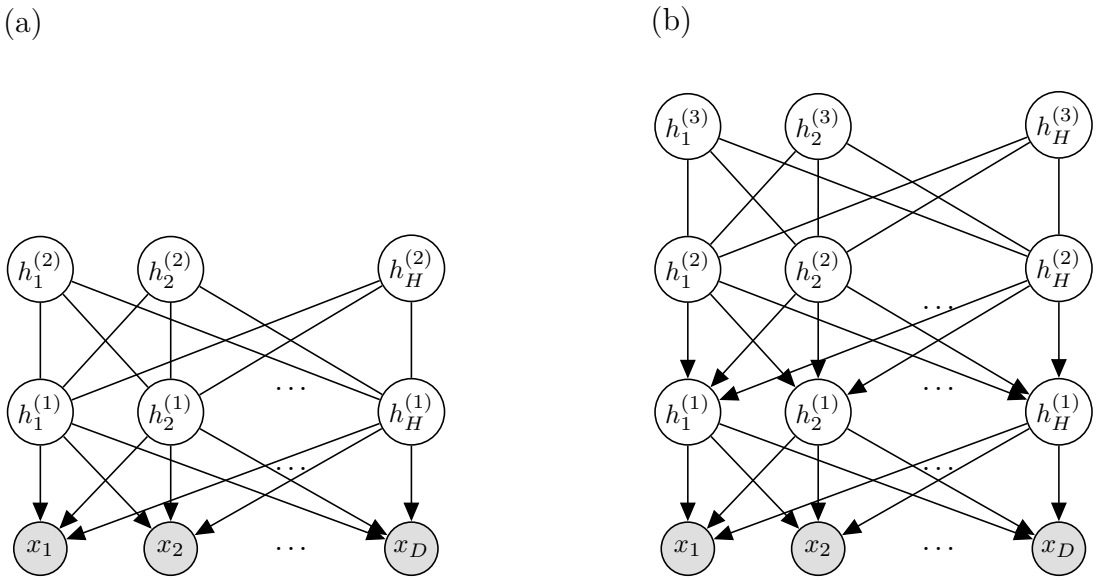

Figure 2.4: Chain-graph graphical model representation of a deep belief network. In (a) a deep belief network with two hidden layers is shown. In (b) a deep belief network with three hidden layers is shown. The marginal distribution of the top two hidden layers is modelled by an RBM, while the rest of variables are modelled by a Bernoulli distribution whose expectation depends only on the configuration of the layer directly above it.

different set of parameters $\boldsymbol{\theta}^{(2)}$:

$$p(\boldsymbol{x}\,|\,\boldsymbol{\theta},\boldsymbol{\theta}^{(2)}) = \sum_{\boldsymbol{h}} p(\boldsymbol{h}\,|\,\boldsymbol{\theta}^{(2)}) p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})\,. \tag{2.55}$$

If this model is to have a higher marginal likelihood for the training data than the RBM, the new $p(\boldsymbol{h}\,|\,\boldsymbol{\theta}^{(2)})$ should better approximate the aggregated posterior (Tang et al., 2012) $\sum_{\boldsymbol{x}} p_{emp}(\boldsymbol{x}) p(\boldsymbol{h}\,|\,\boldsymbol{x},\boldsymbol{\theta})$ (where p_{emp} is the empirical distribution of $\boldsymbol{x}$) more accurately than $p(\boldsymbol{h}\,|\,\boldsymbol{\theta})$ given by the original RBM.

A way of guaranteeing an non-decrease in likelihood is to use another RBM to model $p(\boldsymbol{h})$ (Hinton et al., 2006) adding a new layer of hidden units (see Figure 2.4a). In the binary observables case, this new RBM with H visible units and $H^{(2)}$ hidden units is guaranteed to be able to model $p(\boldsymbol{h})$ at least as well as the original RBM as long as $H^{(2)} \geq D$. Simply by setting $W^{(2)} = W^\top$, $\boldsymbol{b}^{(2)} = \boldsymbol{c}$ and $\boldsymbol{c}^{(2)} = \boldsymbol{b}$ we would obtain the original $p(\boldsymbol{h}\,|\,\boldsymbol{\theta})$; and by optimizing the parameters to better model the empirical $p(\boldsymbol{h}\,|\,\boldsymbol{x},\boldsymbol{\theta})$, an increase in log-likelihood for the training data may be obtainable.

The addition of an extra hidden layer can be repeated as many times as

desired (see Figure 2.4b), always guaranteeing a non-decrease in the log-likelihood obtainable for the training data.

However, this greedy one-layer-at-a-time procedure is suboptimal. Given the availability of an untied distribution over the latent units, a better joint configuration of the parameters of all layers $\boldsymbol{\theta}, \boldsymbol{\theta}^{(2)}, \ldots$ may be found. Hinton et al. (2006) recommend using the greedy algorithm only as an initialization and utilizing the wake-sleep algorithm (see Section 2.6) to obtain a better generative model.

The main use of this DBN training algorithm has been as an initialization for feedforward neural networks. The latent variables in higher layers are hypothesised to be good high level descriptors of the data. However, as commented before, this kind of initialization seems, at present, unnecessary.

Calculating normalized probabilities (or probability densities in the case of continuous observations) under a deep belief networks is intractable. Densities can be approximated using Markov chain Monte Carlo methods (Murray and Salakhutdinov, 2009), but it is computationally costlier than under a restricted Boltzmann machine.

2.4.4 Deep Boltzmann Machines

Deep Boltzmann machines (DBMs) (Salakhutdinov and Hinton, 2009) are a generalization of restricted Boltzmann machines to several layers of latent units. The energy function of a DBM with L hidden layers is defined as:

$$E(\boldsymbol{x}, \boldsymbol{h}) = -{\boldsymbol{b}^{(0)}}^{\top} \boldsymbol{x} - \boldsymbol{x}^{\top} \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - {\boldsymbol{b}^{(1)}}^{\top} \boldsymbol{h}^{(1)} - \sum_{l=2}^{L} \left({\boldsymbol{h}^{(l-1)}}^{\top} \boldsymbol{W}^{(l)} \boldsymbol{h}^{(l)} + {\boldsymbol{b}^{(l)}}^{\top} \boldsymbol{h}^{(l)} \right), \tag{2.56}$$

where $\boldsymbol{h}^{(l)}$ denotes the l-th layer of latent units, and the parameters of the model are the biases, $\boldsymbol{b}^{(0 \ldots L)}$, and weights, $\boldsymbol{W}^{(1 \ldots L)}$, for each layer. Therefore, an RBM can be seen as a special case of the DBM with just one hidden layer.

In general, in a DBM the distribution over latent variables given a configuration of the visible variables does not factorize. Therefore calculating the free-energy of a visible configuration is not tractable. This makes exact calculation of the log-likelihood gradients of a DBM even more computationally intractable than an RBM, as both terms in (2.47) are intractable. Techniques to approximate each term were developed by Salakhutdinov and Hinton (2009), who were able to

successfully train DBMs with several hidden layers. Calculating the log-likelihood of a DBM for a given dataset is also intractable but can be approximated using annealed importance sampling (Murray and Salakhutdinov, 2009; Salakhutdinov and Hinton, 2009).

2.5 Density networks

Following MacKay (1995b), we will refer to directed latent variable models with a single layer of latent variables and a parametric non-linear mapping between latent variables and observations as *density networks*. Note that, despite their name, density networks can model discrete valued data.

Density networks can be seen as a generalization of factor analysis, where the linear mapping between the H-dimensional hidden variables, $\boldsymbol{h}$, and the D-dimensional observations, $\boldsymbol{x}$, is allowed to be non-linear. In a density network model, the distribution over the hidden variables, $p(\boldsymbol{h})$, is set *a priori* (for example, unit spherical Gaussian) and only a parametric model for $p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})$ is learnt. The probability of an observation under a density network is given by:

$$p(\boldsymbol{x}\,|\,\boldsymbol{\theta}) = \int p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})p(\boldsymbol{h})d^H\boldsymbol{h}\,. \tag{2.57}$$

MacKay (1995b) proposed using a tractable parametric form for the observation's distribution given the latent variable, and estimating (2.57) by taking S samples, $\left\{\boldsymbol{h}^{(s)}\right\}_{s=1}^{S}$, from the latent variable's prior distribution $p(\boldsymbol{h})$:

$$p(\boldsymbol{x}\,|\,\boldsymbol{\theta}) \approx \frac{1}{S}\sum_{s=1}^{S} p(\boldsymbol{x}\,|\,\boldsymbol{h}^{(s)},\boldsymbol{\theta})\,. \tag{2.58}$$

Training can be done by stochastic gradient ascent on the log-likelihood of the parameters:

$$\frac{\partial \log p(\boldsymbol{x}\,|\,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} = \frac{1}{p(\boldsymbol{x}\,|\,\boldsymbol{\theta})}\int p(\boldsymbol{h})\frac{\partial p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i}d^H\boldsymbol{h} \tag{2.59}$$

$$= \frac{1}{p(\boldsymbol{x}\,|\,\boldsymbol{\theta})}\int p(\boldsymbol{h})p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})\frac{\partial \log p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i}d^H\boldsymbol{h} \tag{2.60}$$

$$= \frac{1}{\int p(\boldsymbol{h})p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})d^H\boldsymbol{h}}\int p(\boldsymbol{h})p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})\frac{\partial \log p(\boldsymbol{x}\,|\,\boldsymbol{h},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i}d^H\boldsymbol{h}\,, \tag{2.61}$$

where the last expression, can also be approximated by sampling from the prior over the latent variables:

$$\frac{\partial \log p(\boldsymbol{x}\,|\,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} \approx \frac{\sum_{s=1}^{S} p(\boldsymbol{x}\,|\,\boldsymbol{h}^{(s)},\boldsymbol{\theta})\frac{\partial \log p(\boldsymbol{x}\,|\,\boldsymbol{h}^{(s)},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i}}{\sum_{s=1}^{S} p(\boldsymbol{x}\,|\,\boldsymbol{h}^{(s)},\boldsymbol{\theta})}\,. \tag{2.62}$$

These estimators for the density and the log-density gradient with respect to the parameters are unbiased. However, they can have very high variance when the prior over latent variables is very different from the posterior distribution $p(\boldsymbol{h}\,|\,\boldsymbol{x})$; requiring a very high number of samples in order to obtain good estimates. Low-dimensional latent spaces can be sampled in a regular grid, in which case an expectation-maximization algorithm can be used for training (Bishop et al., 1998).

2.6 Helmholtz machines

Helmholtz machines (Hinton et al., 1995; Dayan et al., 1995; Dayan and Hinton, 1996) model data using a hierarchy of L latent variable layers where the joint distribution is given by:

$$p(\boldsymbol{x}, \boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(L)}) = p(\boldsymbol{h}^{(L)}) p(\boldsymbol{h}^{(L-1)}\,|\,\boldsymbol{h}^{(L)}) \ldots p(\boldsymbol{h}^{(1)}\,|\,\boldsymbol{h}^{(2)}) p(\boldsymbol{x}\,|\,\boldsymbol{h}^{(1)})\,. \tag{2.63}$$

Typically, Helmholtz machines further assume a factorial distribution over the variables on each layer conditioned on the value of the layer above, such that

$$p(\boldsymbol{h}^{(L-1)}\,|\,\boldsymbol{h}^{(L)}) = \prod_j p(h_j^{(L-1)}\,|\,\boldsymbol{h}^{(L)})\,. \tag{2.64}$$

A Bayes network graphical model showing the generative model of a Helmholtz machine can be seen in Figure 2.5a. Provided each of the conditionals in (2.64) is tractable, computing the density of a particular configuration of the latent and observed variables can be done exactly and tractably. Also, sampling from these models can be done efficiently using ancestral sampling. However, marginalizing the latent variables to calculate the density of an observation, and learning the parameters from a dataset of observations, is intractable due to the "explaining away" effect in Bayes networks (Pearl, 1988).

A similar approach to density networks, sampling from the prior, can be used. However, sampling from the prior distribution of $p(\boldsymbol{h})$ is, in general, very inefficient and it will lead to an estimator of very high variance (Dayan and Hinton, 1996).

The defining characteristic of Helmholtz machines is the use of a separate tractable recognition model $q(\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(L)}\,|\,\boldsymbol{x})$ to approximate $p(\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(L)}\,|\,\boldsymbol{x})$. Like the generative model, the recognition model also has, typically, a layered configuration, but reverses the roles of dependent and explanatory variables of

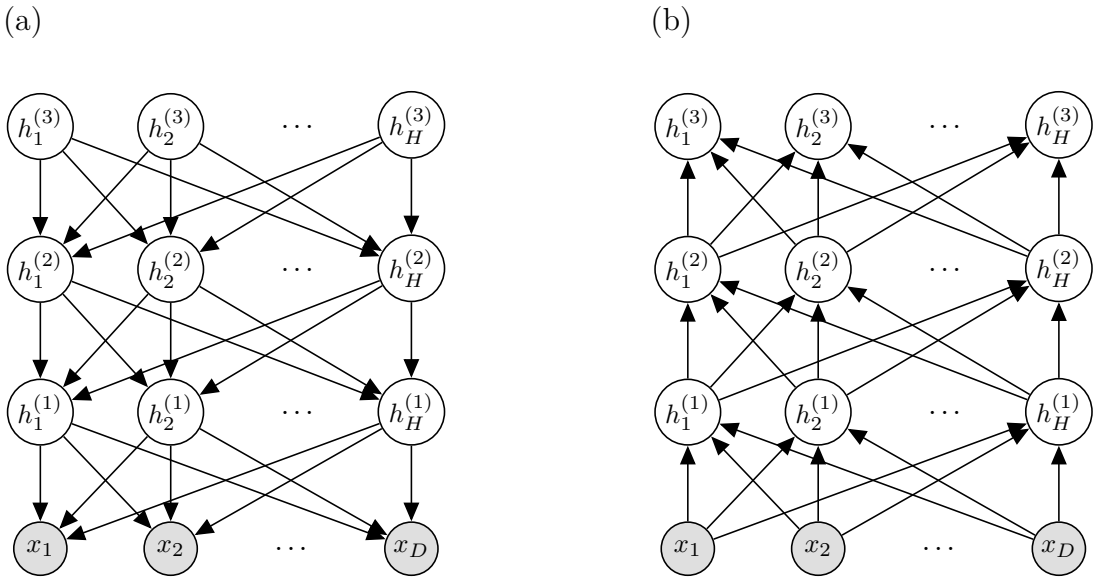

Figure 2.5: Bayes network graphical representation of a three-hidden-layer Helmholtz machine. (a) shows the generative model; (b) shows the recognition model. In both cases conditional independence among units in the same layer is assumed given the layer above in (a) or below in (b).

each conditional distribution:

$$q(\boldsymbol{h}^{(1)},\ldots,\boldsymbol{h}^{(L)}\,|\,\boldsymbol{x}) = q(\boldsymbol{h}^{(1)}\,|\,\boldsymbol{x})q(\boldsymbol{h}^{(2)}\,|\,\boldsymbol{h}^{(1)})\ldots q(\boldsymbol{h}^{(L)}\,|\,\boldsymbol{h}^{(L-1)})\,. \tag{2.65}$$

A factorial distribution for each conditional in the recognition model was assumed in the classical Helmholtz machine publications, even though $p(\boldsymbol{h}^{(l)}\,|\,\boldsymbol{h}^{(l-1)})$ will rarely be factorial. A Bayes network graphical model showing the recognition model of a Helmholtz machine can be seen in Figure 2.5b.

As show in the variational-EM derivations of section 2.3.1, it is possible to obtain a lower bound for the marginal log-density of an observation $\boldsymbol{x}$ under the generative model, p, by calculating the expectation of the joint $p(\boldsymbol{x},\boldsymbol{h})$ when $\boldsymbol{h}$ is distributed following the recognition model q:

$$\log p(\boldsymbol{x}) \geq \left\langle \log p(\boldsymbol{x},\boldsymbol{h}^{(1)},\ldots,\boldsymbol{h}^{(L)}) \right\rangle_{\boldsymbol{h}^{(1)}\ldots\boldsymbol{h}^{(L)}\sim q_{|\,\boldsymbol{x}}} + \left\langle -\log q(\boldsymbol{h}^{(1)},\ldots,\boldsymbol{h}^{(L)}\,|\,\boldsymbol{x}) \right\rangle_{\boldsymbol{h}^{(1)}\ldots\boldsymbol{h}^{(L)}\sim q_{|\,\boldsymbol{x}}}. \tag{2.66}$$

In order for this bound to be close to the actual marginal log-probability of $\boldsymbol{x}$ under p, the gap given by $\mathrm{KL}\,(q \parallel p)$ must be small. To achieve this, Hinton et al. (1995) designed the wake-sleep algorithm that simultaneously trains both models, generative and recognition, by alternating the so called *wake* and *sleep* phases:

Wake phase : A datapoint from the training dataset is used to sample a value for each hidden variable using q. Then the parameters of the generative model are modified to increase the probability of the joint configuration. That is, a stochastic gradient ascent update is performed on the parameters of p to increase the first term on the r.h.s of (2.66)

Sleep phase : A value for every hidden variable and the observations is sampled from the generative model. The parameters of the recognition model are modified to increase the probability of those latent variables being generated by q.

The main criticism[7] towards the wake-sleep training algorithm is that the sleep phase does not correspond to optimising the parameters of q to maximize the bound in (2.66) (Dayan and Hinton, 1996; Mnih and Gregor, 2014; Bornschein and Bengio, 2014). The actual stochastic gradient updates of the parameters of q with respect to the lower bound have very high variance (Dayan and Hinton, 1996). Recently, with the help of several variance reduction techniques, Mnih and Gregor (2014) have designed an estimator of the gradient of the lower bound with respect to the parameters of q whose variance is low enough to be of practical use. Helmholtz machines trained using this estimator of the correct gradient achieved higher log-likelihood than those trained using the wake-sleep algorithm (Mnih and Gregor, 2014).

Another interpretation of the Helmholtz machine poses the recognition model as an importance sampling proposal distribution (MacKay, 1995b; Bornschein and Bengio, 2014). This allows estimating the probability of an observation by using S samples from $q(\boldsymbol{h}\,|\,\boldsymbol{x})$:

$$p(\boldsymbol{x}\,|\,\boldsymbol{\theta}) \approx \frac{1}{S}\sum_{s=1}^{S} \frac{p(\boldsymbol{h}^{(s)})}{q(\boldsymbol{h}^{(s)}\,|\,\boldsymbol{x})} p(\boldsymbol{x}\,|\,\boldsymbol{h}^{(s)}, \boldsymbol{\theta})\,. \tag{2.67}$$

Unfortunately we are usually interested in estimating log-densities, and this estimator is biased if used for that purpose. Nonetheless, it is expected to underestimate the log-likelihood of the model, and therefore offer a safe estimation of its statistical performance.

[7] In the original formulation of the Helmholtz machine, the authors were not only interested in fitting models that obtained high likelihoods, they were also interested in finding a plausible explanation for unsupervised learning in biological neural networks. Therefore they saw the "purely local delta rule" of the wake-sleep algorithm as a necessity (Dayan and Hinton, 1996).

New training techniques that allow for joint training of the generative and recognition models have been recently devised (Gregor et al., 2014; Jimenez-Rezende et al., 2014; Kingma and Welling, 2014). These techniques are able to backpropagate through the stochastic units, and can, therefore, train both networks as a single stochastic autoencoder; where the recognition network acts as the encoder and the generation network as the decoder. These new techniques simplify training and obtained better statistical performance than wake-sleep training.

Helmholtz machines are capable of capturing complex dependencies between the visible variables (Frey et al., 1996; Mnih and Gregor, 2014). Sampling from Helmholtz machines can be done exactly and efficiently. However, even with the use of an inference network, it is not possible to calculate normalized densities exactly, although a stochastic approximation can be obtained by sampling the hidden configuration using q. The interest of the machine learning community in Helmholtz machines has enjoyed a recent revival and constitutes a promising area of research.

2.7 Generalized denoising autoencoders and generative stochastic networks

Generalized denoising autoencoders (GDAE) (Bengio et al., 2013, 2014) take a different approach to modelling complex probability distributions. A GDAE defines a joint probability distribution as the stationary distribution of an ergodic Markov chain. The parameters, $\boldsymbol{\theta}$, of this Markov chain's transition operator are fitted to capture the distribution of interest, $\mathcal{P}$.

In a GDAE the Markov transition operator is given by the sequential sampling of two conditional distributions $C(\tilde{\boldsymbol{x}} \,|\, \boldsymbol{x})$ and $P(\boldsymbol{x} \,|\, \tilde{\boldsymbol{x}}, \boldsymbol{\theta})$:

$$\boldsymbol{x}_t \sim P(\boldsymbol{x} \,|\, \tilde{\boldsymbol{x}}_t, \boldsymbol{\theta}), \quad \text{where} \quad \tilde{\boldsymbol{x}}_t \sim C(\tilde{\boldsymbol{x}} \,|\, \boldsymbol{x}_{t-1}). \tag{2.68}$$

These conditional distributions correspond to the stochastic corruption and denoising conditional distributions of a denoising autoencoder (Vincent et al., 2008). Sampling C corrupts a particle $\boldsymbol{x}_{t-1}$ (e.g. adds a Gaussian distributed random vector), while P attempts to recover its value, moving the particle back to the region of high probability density.

If certain constraints for $\mathcal{P}$, C and P are satisfied[8], then the stationary distribution of the Markov chain defined by C and P will be consistent with $\mathcal{P}$ (Bengio et al., 2013).

The intuition behind GDAE is that, for some distributions, it is easier to model the local structure, $\mathcal{P}(\boldsymbol{x}\,|\,\tilde{\boldsymbol{x}})$ than to model the distribution of interest $\mathcal{P}(\boldsymbol{x})$.

Generative stochastic networks (GSN) generalize GDAEs by introducing a latent variable in the Markov chain such that:

$$\boldsymbol{x}_t \sim P(\boldsymbol{x}\,|\,\boldsymbol{h}_t, \boldsymbol{\theta}), \quad \text{where} \quad \boldsymbol{h}_t \sim E(\boldsymbol{h}\,|\,\boldsymbol{h}_{t-1}, \boldsymbol{x}_{t-1}). \tag{2.69}$$

That is, the corrupted particles of GDAEs, $\tilde{\boldsymbol{x}}$ are substituted by $\boldsymbol{h}$, a stochastic encoding of $\boldsymbol{x}$ that can take values in a different domain from that of $\boldsymbol{x}$. By using the right, problem dependent, latent encoding, the GSN can result in a Markov chain with faster mixing.

Approximate samples from a GDAE or a GSN can be obtained by sampling from the Markov chain (after an initial burn-in period). Normalized densities can be approximated by using a kernel density-estimator on a sufficient number of samples. Inference, i.e. sampling or density estimation conditioned on a certain value for a subset of dimensions, can be done by fixing those variables while running the Markov chain.

GDAEs and GSN that utilize neural networks to model the Markov transition operator can model complex distributions (Bengio et al., 2013, 2014). Their main practical disadvantage is the need to run a Markov chain in order to do inference, which, depending on the number of samples utilized and the length of the burn-in period, may be computationally expensive.

2.8 Autoregressive models

We will adopt Frey's definition of an autoregressive model (Frey, 1998), as a Bayesian network with no latent variables and cascade connectivity. That is, the distribution of each variable depends on all other variables preceding it in a given ordering, o, of the variables. Formally, o is a D-tuple holding a permutation of the visible variable indices $1 \ldots D$. The Bayes network of an autoregressive model is fully specified by choosing an ordering, o, of the variables, as shown in Figure 2.6.

[8] The most restrictive of these constraints is the consistency of the reconstruction conditional $P(\boldsymbol{x}\,|\,\tilde{\boldsymbol{x}}, \boldsymbol{\theta})$ with the true $\mathcal{P}(\boldsymbol{x}\,|\,\tilde{\boldsymbol{x}}) \propto C(\tilde{\boldsymbol{x}}\,|\,\boldsymbol{x})\mathcal{P}(\boldsymbol{x})$

Figure 2.6: Bayes' network graphical model of two autoregressive networks specified by different orderings of the variables. In (a) $o = [1,2,3,4]$, in (b) $o = [4,3,2,1]$. All nodes are visible and direct descendants of all nodes preceding them.

General autoregressive models assume no conditional independences. Their Bayes network specifies a particular factorisation of the probability density function:

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}), \tag{2.70}$$

where $\boldsymbol{x}_{o_{<d}}$ stands for the subset of visible variables indexed by the first $d-1$ elements of o.

Given the lack of conditional independence assumptions, any probability distribution can be modelled using an autoregressive model. However, an inductive bias is introduced by choosing a parametric form for each of the conditionals in (2.70) and an ordering of the variables.

Density estimation under autoregressive models has mild computational complexity, but some inference tasks require approximate methods. If each of the one-dimensional conditionals in (2.70) is tractable, it is possible to calculate the joint density, $p(\boldsymbol{x})$, tractably simply by multiplying the value of each conditional. Marginalising out variables at the end of the ordering, o, can be done efficiently, simply ignoring the conditionals for those variables in (2.70), as they add up to 1:

$$p(x_{o_1}, x_{o_2}, \ldots, x_{o_k}) = \sum_{x_{o_{k+1}}, x_{o_{k+2}} \ldots x_{o_D}} \left(\prod_{d=1}^{D} p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}) \right) \tag{2.71}$$

$$= \prod_{d=1}^{k} p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}) \underbrace{\sum_{x_{o_{k+1}}, x_{o_{k+2}} \ldots x_{o_D}} \left(\prod_{d=k+1}^{D} p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}) \right)}_{=1} \tag{2.72}$$

$$= \prod_{d=1}^{k} p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}) \,. \tag{2.73}$$

However, marginalising out a subset of the dimensions that is not at the end of o involves summing a number of terms exponential in the number of missing

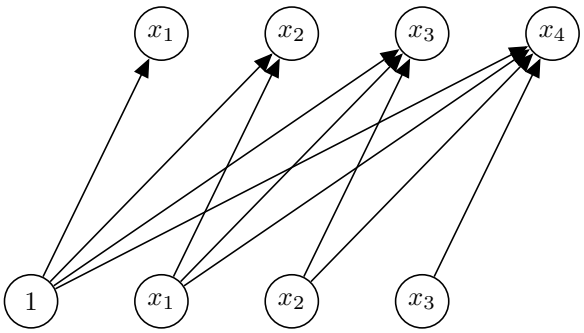

Figure 2.7: Neural network diagram (note the lack of hidden layers) that implements a logistic autoregressive Bayesian network (this is not a Bayes network graphical model). The restricted connectivity results in a triangular weight matrix. The node labelled "1" is the bias. Each variable is predicted by a logistic regressor that uses as predictors all variables preceding it.

dimensions, and will usually require approximate inference techniques like MCMC, and variational methods (e.g. Bishop, 2006).

In the rest of this section we review some autoregressive distribution estimators that have appeared in the literature in the last twenty years. Without loss of generality we will assume the natural index order, $o = 1, 2, \ldots, D$ of the dimensions in the following sections. The models can be defined for any other ordering simply by permuting the dimensions.

2.8.1 Logistic autoregressive Bayesian networks

Logistic autoregressive Bayesian networks (LARN)[9] are the earliest example of autoregressive methods for joint probability estimation (Frey, 1998). A LARN models a multidimensional binary variable $\boldsymbol{x} \in \{0,1\}^D$ by using a logistic regressor to model each one-dimensional conditional in (2.70). That is, the probability of each variable taking value 1 is modelled by an affine transformation of the variables preceding it in the ordering.

$$P(x_d = 1 \,|\, \boldsymbol{x}_{<d}) = \text{sigm}\left(\boldsymbol{w}^{(d)} \boldsymbol{x}_{<d} + b_d\right), \tag{2.74}$$

where the model parameters $\boldsymbol{w}^{(d)}$ belongs to $\mathbb{R}^{d-1}$ and b_d to $\mathbb{R}$. A graphical representation of the neural network that implements a LARN can be seen in Figure 2.7.

[9]LARNs are also called fully visible Bayes networks (FVBN).

LARNs can only capture first-order dependencies between the variables. The fact that a predictor takes value 1, will always increase or decrease the probability of the variable being predicted taking value 1, independently of the value of the rest of the predictors. That is, a LARN cannot model interactions between predictor variables.

Calculating the probability mass of an observation has complexity $O(D^2)$, which is tractable. The number of parameters of the model also grows quadratically with the dimensionality of the data. This may make it necessary to use Bayesian methods or regularisation techniques, like weight sparsity or weight decay, in order to avoid overfitting to small datasets. However, for most datasets LARN's main drawback will be its limited representational power.

The methods described next address the two limitations of LARNs. Namely, their inability to capture complex interactions between the predictor variables in each conditional, and the fixed number of parameters.

2.8.2 Autoregressive neural network models

The use of neural networks with a hidden layer to implement the autoregressive conditionals in (2.70) was analysed by Bengio and Bengio (2000). These autoregressive neural network models (ANNM) address the main limitation of LARNs, namely their inability to capture interactions between predictor variables, by adding a layer of non-linear hidden features.

The architecture proposed by Bengio and Bengio can be seen as multiple neural networks with tied parameters. The value of hidden units that have been used to predict the preceding dimensions in the ordering, o, are reused. Only K_H new hidden units with full connectivity to the dimensions already predicted are added to each network, as shown in Figure 2.8.

The model can also be interpreted as a single neural network that predicts the distribution over all dimensions simultaneously, but guarantees a valid implementation of the factorisation in (2.70) by restricting the connectivity of the output probability unit for the d-th dimension, to the first $K_H \times d$ hidden units; and from those hidden units to the input value of the first $d-1$ dimensions, as shown in Figure 2.9.

Formally, for a binary dataset, the distribution over the d-th dimension in an

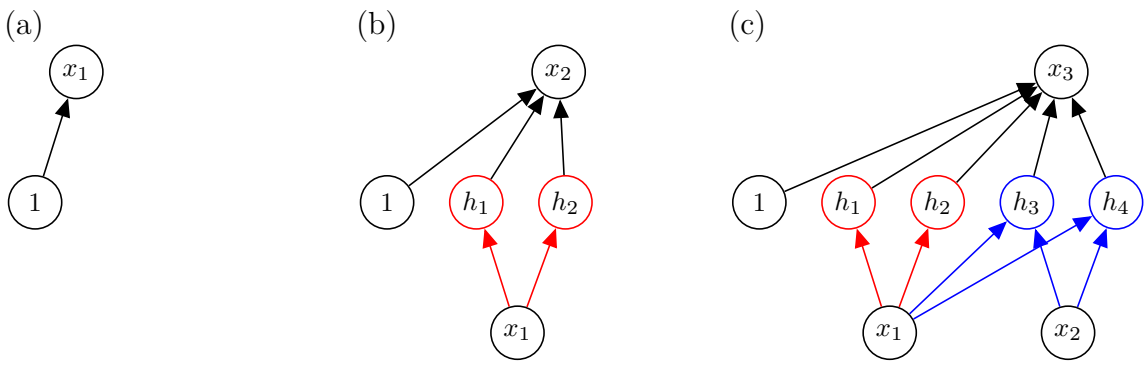

Figure 2.8: Neural network diagram (this is not a Bayes network graphical model) of an autoregressive neural network model with three-dimensional observations and $K_H = 2$. Nodes labelled "1" are biases (input bias not shown for clarity). The probability distribution of each dimension is calculated using the value of dimensions that precede it. (a), (b) and (c) show the networks predicting the first, second and third dimensions in the ordering $o = 1, 2, 3$. Weights with the same colour are shared across networks. Note the restricted connectivity, caused by the addition of K_H new hidden units with each new input. An ANNM can also be represented as a single network, see Figure 2.9

.

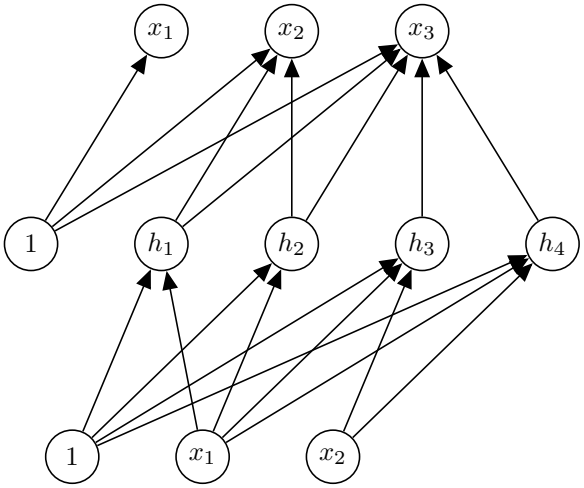

Figure 2.9: Neural network diagram (this is not a Bayes network graphical model) of an autoregressive neural network model with three-dimensional observations and $K_H = 2$. Nodes labelled "1" are biases. Note the restricted connectivity. The directed-paths from inputs to outputs only lead to dimensions that appear later in the ordering o.

observation is calculated as follows:

$$p(x_d = 1 \,|\, \boldsymbol{x}_{<d}) = \text{sigm}\left(\sum_{j=1}^{K_H(d-1)} W_{d,j} h_j + b_d \right), \tag{2.75}$$

$$h_j = \tanh\left(\sum_{i=1}^{j//K_H+1} V_{j,i} x_i + c_j \right), \tag{2.76}$$

where $\cdot//\cdot$ stands for integer division, with weight matrices $\boldsymbol{W} \in \mathbb{R}^{D \times K_H D}$, $\boldsymbol{V} \in \mathbb{R}^{K_H D \times D}$, bias vectors $\boldsymbol{b} \in \mathbb{R}^D$, $\boldsymbol{c} \in \mathbb{R}^{K_H D}$. Note that due to the restricted connectivity, many elements in the W and V matrices are not used. In practice, it can be easily and efficiently implemented using matrix multiplications by setting to zero the weight of the connections not allowed.

The complexity of calculating the probability of an observation under this model is $O(K_H D^2)$, where K_H is the number of hidden units added with each extra predictor variable and D is the dimensionality of the data.

Bengio and Bengio obtained better statistical performance, as measured by the model likelihood on a held out dataset, than LARN on a battery of datasets of small dimensionality.

They also found it advantageous to reduce the number of free parameters by further pruning the connectivity between the inputs and hidden units. In their particular implementation, connections from input j to the hidden units added to predict dimension i are banned if the Kolmogorov-Smirnov statistic between dimensions i and j is lower than a threshold selected using crossvalidation.

2.8.3 Neural autoregressive distribution estimators

The Neural autoregressive distribution estimator (NADE) (Larochelle and Murray, 2011; Gregor and LeCun, 2011) is an autoregressive model of multivariate binary data. NADE is based on single-hidden-layer neural networks, but have a different parameter tying strategy to ANNMs.

Larochelle and Murray developed NADE inspired by the mean-field variational approximation to the autoregressive conditionals of a binary RBM (Welling and Hinton, 2002)[10]. The marginal distribution over visible variables modelled by an RBM can be factorised (just as any distribution) into an autoregressive product

[10] Gregor and LeCun independently designed a model identical to NADE from the point of view of minimum description length (Gregor and LeCun, 2011).

of one-dimensional conditionals:

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_d \,|\, \boldsymbol{x}_{<d}), \tag{2.77}$$

where, given the energy formulation of RBMs, each factor can be rewritten as:

$$p(x_d \,|\, \boldsymbol{x}_{<d}) = \frac{p(x_d, \boldsymbol{x}_{<d})}{p(\boldsymbol{x}_{<d})} \tag{2.78}$$

$$= \frac{\sum_{\boldsymbol{x}_{>d}} \sum_{\boldsymbol{h}} \exp\{-E(\boldsymbol{x}, \boldsymbol{h})\}}{\sum_{\boldsymbol{x}_{\geq d}} \sum_{\boldsymbol{h}} \exp\{-E(\boldsymbol{x}, \boldsymbol{h})\}}. \tag{2.79}$$

However, most of these conditionals are intractable, as they require summing over an exponential number of visible unit configurations. To tackle this problem Larochelle and Murray calculate the mean-field variational approximation to the distribution $p(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d})$ from which they can easily approximate $p(x_d \,|\, \boldsymbol{x}_{<d})$ due to the factorial nature of the mean-field approximation. In more detail, they fit the factorial distribution:

$$q(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d}) = \prod_{j=d}^{D} {\mu_j}^{v_j} (1-\mu_j)^{(1-v_j)} \prod_{k=1}^{H} \tau_k^{h_k} (1-\tau_k)^{(1-h_k)}, \tag{2.80}$$

where μ_j and τ_k are the parameters of the variational model, corresponding to the expected value of x_j and h_k respectively. In order to fit those parameters, the KL-divergence between $q(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d})$ and $p(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d})$ is minimized. The following update equations, which are iterated to convergence, are obtained:

$$\tau_k = \text{sigm}\left(\sum_{j=1}^{d-1} W_{k,j} x_j + \sum_{j=d}^{D} W_{k,j} \mu_j + c_k \right), \tag{2.81}$$

$$\mu_j = \text{sigm}\left(\sum_{k=1}^{H} W_{k,j} \tau_k + b_j \right). \tag{2.82}$$

Once convergence has been achieved, the marginal $q(x_d = 1 \,|\, \boldsymbol{x}_{<d})$ can be easily calculated as μ_d. Approximating each $p(x_d \,|\, \boldsymbol{x}_{<d})$ requires fitting a different mean-field approximation, each of which may need tens of mean-field iterations.

Based on the update equations (2.81) and (2.82), Larochelle and Murray developed NADE, based on a single mean-field update, which they treat as a model on its own right, instead of an approximation to an RBM. Therefore, assuming that μ_j is initialized to zero, we would obtain an autoregressive model, where the distribution of each one-dimensional conditional is given by:

$$p(x_d = 1 \,|\, \boldsymbol{x}_{<d}) = \text{sigm}\left(\boldsymbol{W}_{\cdot,d}^{\top} \boldsymbol{h}(d) + b_d \right) \tag{2.83}$$

$$h_k(d) = \text{sigm}\left(\boldsymbol{W}_{k,<d} \boldsymbol{x}_{<d} + c_k \right), \tag{2.84}$$

where the values of the hidden variables is different for each conditional, thus their dependence on d. The model is parameterised by $\boldsymbol{W} \in \mathbb{R}^{H \times D}$, $\boldsymbol{b} \in \mathbb{R}^{D}$, $\boldsymbol{c} \in \mathbb{R}^{H}$, which are shared by all conditionals.

However, better results can be obtained by untying the weight matrices used to calculate the hidden unit values and the conditional probabilities. This results in the definitive formulation of NADE:

$$p(x_d = 1 \,|\, \boldsymbol{x}_{<d}) = \operatorname{sigm}\left(\boldsymbol{V}_{\cdot,d}^{\top} \boldsymbol{h}(d) + b_d\right), \quad \text{where} \tag{2.85}$$

$$h_k(d) = \operatorname{sigm}\left(\boldsymbol{W}_{k,<d} \boldsymbol{x}_{<d} + c_k\right) \tag{2.86}$$

with parameters $\boldsymbol{W} \in \mathbb{R}^{H \times D-1}$, $\boldsymbol{V} \in \mathbb{R}^{H \times D}$, $\boldsymbol{b} \in \mathbb{R}^{D}$, $\boldsymbol{c} \in \mathbb{R}^{H}$.

The weights between the inputs and the hidden units for the neural networks that calculate each conditional are tied: $\boldsymbol{W}_{\cdot,<d}$ stands for the first $d-1$ columns of a shared weight matrix $\boldsymbol{W}$, see Figure 2.10. This parameter sharing reduces the total number of parameters to quadratic in the number of input dimensions (assuming that H is proportional to D); lessening the need for regularisation. Computing the probability of a datapoint can be done with complexity quadratic with the dimensionality of the data, $O(DH)$, by sharing the computation when calculating the hidden activation of each neural network ($\boldsymbol{W}_{\cdot,<d} \boldsymbol{x}_{<d} + \boldsymbol{c}$) recursively. This only requires the multiplication of a vector times a scalar, and the addition of two H-dimensional vectors:

$$\boldsymbol{a}(1) = \boldsymbol{c} \tag{2.87}$$

$$\boldsymbol{a}(d+1) = \boldsymbol{a}(d) + \boldsymbol{W}_{\cdot,d} x_d \tag{2.88}$$

This mild computational complexity makes it possible to train a NADE using gradient ascent techniques. It is also possible to obtain exact independent samples from the model with complexity $O(DH)$ using ancestral sampling. The likelihood of a NADE for a dataset can also be calculated efficiently, and thus its statistical performance can be compared to that of other models.

In their experiments, conducted on binary images of handwritten digits (LeCun et al., 1998) and a collection of UCI datasets (Bache and Lichman, 2013), Larochelle and Murray found NADE obtained likelihoods comparable to intractable RBMs (whose likelihoods had been approximated using annealed importance sampling). The results were much superior to the likelihoods of other tractable models (small RBMs and mixtures of multivariate Bernoullis).

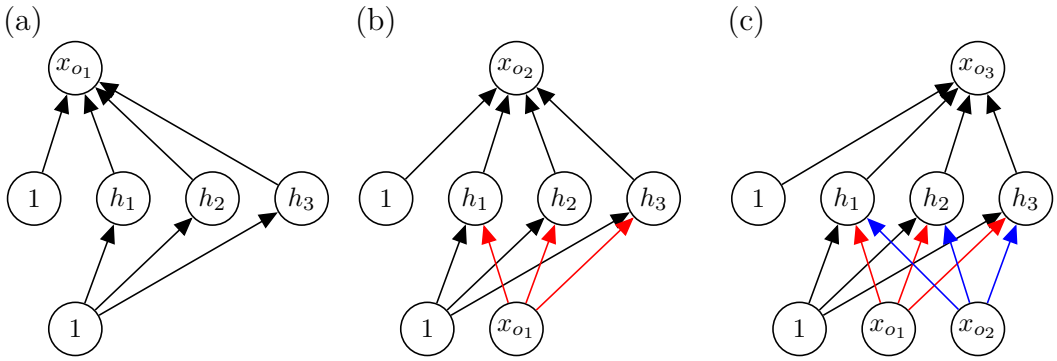

Figure 2.10: Neural network diagram (this is not a Bayes network graphical model) of a NADE model with three-dimensional observations. Nodes labelled "1" are biases. The probability distribution of each dimension is calculated using the value of dimensions that precede it in the ordering o. (a), (b) and (c) show the networks predicting the first, second and third dimensions in the ordering o. Weights with the same colour are shared across networks. Unlike in ANNMs, when an input is present, it is connected to every hidden unit.

In contrast to ANNMs, NADE uses the same number of hidden units to calculate each conditional distribution. Both models can be interpreted as a series of artificial neural networks (one for each one-dimensional conditional) that share some of their parameters. NADE's main advantage is its better computational complexity $O(HD)$ versus ANNM's $O(K_H D^2)$. However if H is approximately $K_H D$ then both models have similar computational requirements.

NADE has been extended to allow the modelling of count data by Larochelle and Lauly (2012). However no real-valued version of the model had been developed before the investigations presented in this thesis.

Chapter 3

The real-valued neural autoregressive density estimator

This chapter is an extended version of the article "RNADE: The real-valued neural autoregressive density estimation" (Uria et al., 2013) published in NIPS 2013.

In this chapter we introduce RNADE, a new model for joint density estimation of real-valued vectors (Sections 3.2, 3.3). Our model calculates the density of a datapoint as the product of one-dimensional conditionals modelled using mixture density networks with shared parameters. RNADE learns a distributed representation of the data, while having a tractable expression for the calculation of densities. A tractable likelihood allows direct comparison with other methods and training by standard gradient-based optimizers. We compare the performance of RNADE on several datasets of heterogeneous and perceptual data (Section 3.4), finding it outperforms mixture models in all cases. We also investigate its sensitivity to the ordering of the dimensions (Section 3.5).

3.1 Introduction

In this chapter we introduce an extension of NADE able to model real-valued datasets. Real-valued datasets are commonplace. In this thesis we focus on modelling perceptual data, like images and sounds, which is central to artificial intelligence tasks including object and speech recognition. However, modelling of multivariate real-valued data is an important task in many fields. In finance, modelling the dependencies among the returns of different instruments plays a crucial role in portfolio design (Markowitz, 1952). In physics, modelling experi-

mental data can pinpoint anomalies that may suggest better experimental designs or new theoretical developments.

Alas, some of the generative models we reviewed in the previous chapter are only suitable for binary data. Gaussian-RBMs with parameterized conditional standard deviations are difficult to train, and fixing them leads to models with low statistical performance (Theis et al., 2011). The mean-covariance RBM and the spike-and-slab RBM model interesting real-valued datasets better, but suffer from intractable density calculations.

Autoregressive models that utilize neural networks have, to the best of our knowledge, only been studied with binary and discrete-valued data. An extension of these models to real-valued data was suggested before by Bengio and Bengio (2000), who proposed using a network that outputs the parameters of a fixed parametric model for each one-dimensional conditional. However, the implementation or empirical analysis of their statistical performance on interesting real-valued datasets has not been reported.

In the next sections, we extend the formulation of NADE to real-valued data, a model we call the *real-valued neural autoregressive density estimator* (RNADE). Following the development of NADE, we will start by calculating the mean-field update equations of a Gaussian-RBM. However, these equations will suggest a model handicapped by homoscedastic conditionals. We will refine this model by using heteroscedastic conditionals, and then generalize it using mixture density networks to capture multimodal conditionals.

3.2 Gaussian-RBM autoregressive mean-field updates

As we saw in Section 2.8.3, NADE was inspired by the update equations that result from calculating the mean-field approximation to the autoregressive conditionals of an RBM. Similarly, given that the Gaussian-RBM is a real-valued extension of the RBM, we will calculate the mean-field updates of its autoregressive conditionals and establish whether we can derive a model suitable for real-valued data.

We are interested in approximating the $p(x_d \,|\, \boldsymbol{x}_{<d})$ conditionals of a Gaussian-RBM parameterized by: $\boldsymbol{b} \in \mathbb{R}^D$, the visible unit biases, $\boldsymbol{c} \in \mathbb{R}^H$, the latent variable biases, $\boldsymbol{W} \in \mathbb{R}^{D \times H}$, the weights, and $\boldsymbol{\sigma} \in \mathbb{R}^{+D}$, the visible unit conditional standard deviations. To do so, we will calculate the mean-field approximation of $p(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d})$ by a factorial $q(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d})$ from which it is trivial to marginalize

$q(x_d \,|\, \boldsymbol{x}_{<d})$. In this mean-field approximation, the distribution of each visible unit is assumed to take a Gaussian distribution with mean μ_i and standard deviation δ_i; while each latent variable takes a Bernoulli distribution with mean τ_i. The full derivations are shown in Appendix A; here we only show the resulting mean-field update equations:

$$\tau_i = \text{sigm}\left(c_i + \sum_{j=1}^{d-1} \frac{1}{\sigma_j} x_j W_{j,i} + \sum_{j=d}^{D} \frac{1}{\sigma_j} \mu_j W_{j,i}\right), \tag{3.1}$$

$$\mu_i = b_i + \delta_i \sum_{k=1}^{H} W_{i,k} \tau_k, \tag{3.2}$$

$$\delta_i = \sigma_i, \tag{3.3}$$

which should be iterated until convergence. However, if we limit ourselves to a single update, by first updating the parameters of the hidden units (every μ_i is initialized to zero) followed by an update of the visible unit's parameters, we obtain:

$$\tau_i = \text{sigm}\left(c_i + \sum_{j=1}^{d-1} \frac{1}{\sigma_j} x_j W_{j,i}\right), \tag{3.4}$$

$$\mu_i = b_i + \delta_i \sum_{k=1}^{H} W_{i,k} \tau_k, \tag{3.5}$$

$$\delta_i = \sigma_i. \tag{3.6}$$

As in the binary-NADE formulation, we derive an autoregressive model over the visible variables based on the conditionals $q(x_d \,|\, \boldsymbol{x}_{<d})$ given by this single-step mean-field update. Also following the binary-NADE, the weight parameters $\boldsymbol{W}$ in (3.4) and (3.5) are untied into $\boldsymbol{W}$ and a new matrix $\boldsymbol{V}$. This results in an autoregressive model where each one-dimensional conditional is given by:

$$q(x_d \,|\, \boldsymbol{x}_{<d}) = \mathcal{N}(x_d; \mu_d, \sigma_d^2), \tag{3.7}$$

with:

$$\mu_d = b_d + \sigma_d \sum_{k=1}^{H} V_{d,k} h_k, \tag{3.8}$$

$$h_i = \text{sigm}\left(c_i + \sum_{j=1}^{d-1} \frac{1}{\sigma_j} x_j W_{j,i}\right), \tag{3.9}$$

where $\boldsymbol{b} \in \mathbb{R}^D$, $\boldsymbol{c} \in \mathbb{R}^H$, $\boldsymbol{W} \in \mathbb{R}^{D \times H}$, $\boldsymbol{V} \in \mathbb{R}^{D \times H}$, and $\boldsymbol{\sigma} \in \mathbb{R}^{+D}$ are the parameters of the model, which are fitted on their own right to maximize the likelihood of the model.

Once the weight matrices have been untied, the standard deviation factors in (3.8) and (3.9) can be incorporated into the weight matrices $\boldsymbol{W}$ and $\boldsymbol{V}$, finally obtaining a set of equations where the mean of each autoregressive conditional is calculated by a regression artificial neural network:

$$\mu_d = b_d + \sum_{k=1}^{H} V_{d,k} h_k \,, \tag{3.10}$$

$$h_i = \text{sigm}\left(c_i + \sum_{j=1}^{d-1} x_j W_{j,i} \right) . \tag{3.11}$$

The variance of each dimension, σ_d, is fixed. We will call this model RNADE-FV (for fixed variance) in our tables of results.

The fixed variance of each of these autoregressive conditionals is a property inherited from the Gaussian-RBM[1]. It is, nonetheless, a very limiting characteristic: the entropy of the conditional will be the same regardless of the possibility of making more accurate predictions given certain values of the preceding dimensions. In Figure 3.1b we show an example of a distribution where the entropy of x_2 varies with the value of x_1.

Another limitation of this autoregressive model is the unimodal nature of the conditionals (3.7). This will prove problematic when modelling datasets like the one shown in Figure 3.1c. Maximum likelihood training will result in a high variance Gaussian that covers both modes, and possibly has its mode in a region of very low actual probability density. We note that this is not a characteristic inherited from the Gaussian-RBM, whose actual autoregressive conditionals, unlike their mean-field approximations, may be multimodal.

Fitting the variance parameters in the Gaussian-RBM is difficult due to their presence in the numerator of (2.52) and denominator of (2.53)[2]. We expect no difficulty in fitting the standard deviation parameters of an RNADE-FV. The only precaution we take in our implementation is to fit the log-standard-deviation so we can perform unconstrained optimization of all parameters.

[1] In a Gaussian-RBM, the visible units follow independent Gaussian distributions conditioned on the value of the latent units. The variances of these Gaussian do not depend on the value of the latent units.

[2] Small standard deviations make the hidden units saturate unless the weights are very small, in which case the mean of the visible units conditioned on the value of the hidden units will be very close together. That is, it is difficult to learn multimodal distributions. The work of Cho et al. (2011) reformulates the Gaussian-RBM to avoid the presence of σ in the denominator of (2.53).

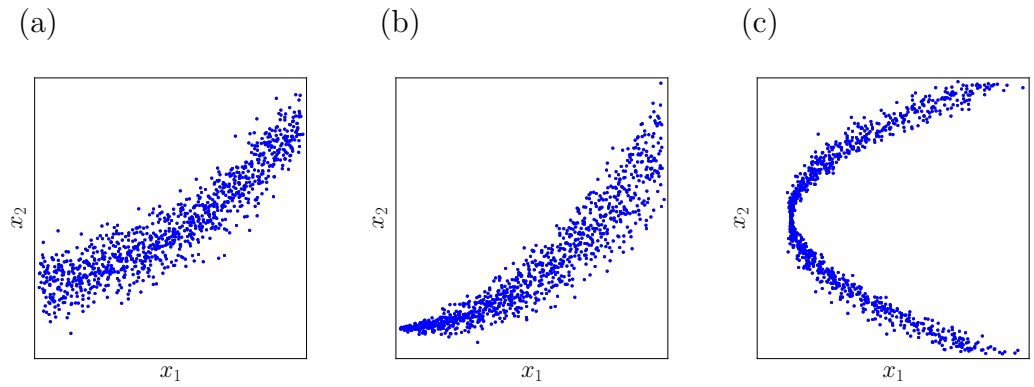

Figure 3.1: Each subfigure shows a thousand samples from a distribution with a different type of $p(x_2 \mid x_1)$ conditional. The conditional is: homocedastic in (a), heterocedastic in (b), and multimodal in (c).

3.3 The real-valued neural autoregressive density estimator

Each of the autoregressive one-dimensional conditionals given by (3.7, 3.10, 3.11) can be seen as a trivial mixture density network (MDN). A mixture density network (Bishop, 1994) is a conditional probabilistic model where the parameters of a mixture of Gaussians are calculated by a neural network that has for input the predictor variables; see Figure 3.2. Mixture density networks are capable of capturing multimodal conditional distributions (e.g. Uria, 2011; Richmond et al., 2003). The conditional distribution modelled by an MDN can be multivariate; in which case, a diagonal covariance matrix is commonly assumed[3]. In order to guarantee that all the variances are positive, the log-standard-deviation is commonly predicted. Similarly, for the component weights the network outputs the energies of a *softmax* distribution in order to guarantee that they are positive and add up to one.

In this light, we can extend the formulation presented in the previous section by substituting the regression neural network used to calculate the mean of each Gaussian with an MDN that outputs the parameters of a mixture of Gaussians. Thus, our model, which we call the *Real-valued Neural Autoregressive Density-*

[3]An investigation on the use of full covariance matrices for low-dimensional data (< 10 dimensions) can be found in Williams (1996).

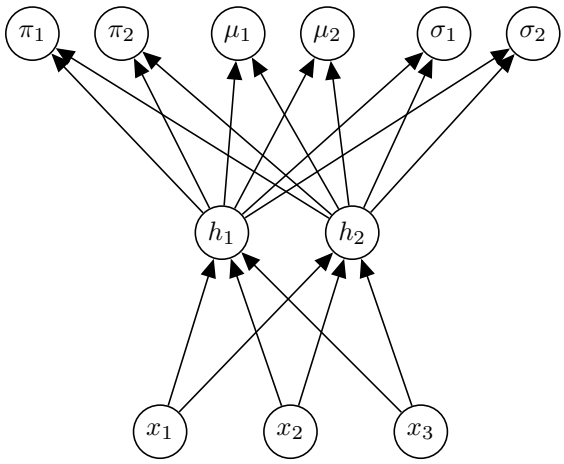

Figure 3.2: Graphical representation of a mixture density network. This particular MDN models a univariate conditional as a mixture of two Gaussian components whose parameters are calculated from three predictor variables using a neural network with two hidden units.

Estimator, or RNADE, models the probability density of a vector $\boldsymbol{x}$ as:

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_d \,|\, \boldsymbol{x}_{<d}) \,. \tag{3.12}$$

Where each of the conditionals is modelled by a mixture of Gaussians:

$$p(x_d \,|\, \boldsymbol{x}_{<d}) = \sum_{c=1}^{C} \pi_{d,c} \mathcal{N}(\mu_{d,c}, \sigma_{d,c}^2) \,. \tag{3.13}$$

Whose parameters are calculated by a regression neural network that has $\boldsymbol{x}_{<d}$ for

inputs:

$$\pi_{d,c} = \frac{\exp\left\{z^{(\pi)}_{d,c}\right\}}{\sum_{c=1}^{C}\exp\left\{z^{(\pi)}_{d,c}\right\}}, \tag{3.14}$$

$$\mu_{d,c} = z^{(\mu)}_{d,c}, \tag{3.15}$$

$$\sigma_{d,c} = \exp\left\{z^{(\sigma)}_{d,c}\right\}, \tag{3.16}$$

$$z^{(\pi)}_{d,c} = b^{(\pi)}_{d,c} + \sum_{k=1}^{H} V^{(\pi)}_{d,k,c} h_{d,k}, \tag{3.17}$$

$$z^{(\mu)}_{d,c} = b^{(\mu)}_{d,c} + \sum_{k=1}^{H} V^{(\mu)}_{d,k,c} h_{d,k}, \tag{3.18}$$

$$z^{(\sigma)}_{d,c} = b^{(\sigma)}_{d,c} + \sum_{k=1}^{H} V^{(\sigma)}_{d,k,c} h_{d,k}, \tag{3.19}$$

$$h_{d,i} = \mathrm{sigm}\left(c_i + \sum_{j=1}^{d-1} x_j W_{j,i}\right). \tag{3.20}$$

As a result, the limitations in RNADE-FV caused by the unimodal fixed-variance conditionals are surmounted. Given that we are only interested in modelling one-dimensional conditional distributions, the usual diagonal covariance assumption of MDNs does not impose any limitations. We will let the autoregressive nature of our model deal with the dependencies between dimensions.

3.3.1 Computational cost of RNADE

As in the binary-NADE model, RNADE can compute the hidden unit activations recursively:

$$h_{d,i} = \mathrm{sigm}\left(a_{d,i}\right), \tag{3.21}$$

$$a_{1,i} = c_i, \tag{3.22}$$

$$a_{d,i} = a_{d-1,i} + x_{d-1} W_{d-1,i}. \tag{3.23}$$

Therefore, RNADE inherits all the mild computational expense characteristics of NADE. Both, density estimation and sampling can be done in quadratic time $O(DH)$ (assuming the number of hidden units is proportional to D the dimensionality of the data). More exactly, in $O(DHC)$ where C stands for the number of components in each one-dimensional conditional, but we can assume C to be a small constant.

3.3.2 Variants of parametric conditionals

Using a mixture of Gaussians to represent the conditional distributions in RNADE is an arbitrary parametric choice. Given several components, a mixture of Gaussians model can represent a rich set of skewed, and multimodal distributions with different tail behaviours. However, other choices may be more appropriate in particular circumstances.

Conditional distributions of perceptual data are often assumed to be Laplacian (e.g. Robinson, 1994). Work on natural images often uses scale mixtures (e.g. Theis et al., 2012), i.e. a mixture of Gaussians where all components share a common mean parameter. Scale mixtures can model kurtotic distribution and have less parameters than a regular mixture of Gaussian, but assume a symmetric unimodal distribution.

A practical disadvantage of mixture model conditionals is the need to cross-validate the number of components used. Moreover, mixture models suffer from aliasing, i.e. several parameter configurations (permutations of the components) produce the same probability distribution. Skewed or kurtotic conditionals can be modelled by skew-Gaussian or t-Student distributions respectively; which have only 3 parameters and do not suffer from aliasing. Both, kurtosis and skew can be captured, for example, by the sinh-arcsinh distribution (Jones and Pewsey, 2009).

We recommend the use of the simplest conditional form possible. The use of conditional forms with a greater number of parameters should be justified by higher likelihoods on held out data, or substantial increases in the training likelihood. To avoid overfitting, the training likelihood should be penalized using, for example, the Bayes' information criterion or the Akaike information criterion (e.g. Barber, 2012).

In section 3.4, we experiment with several variants of one-dimensional conditional: Gaussian (RNADE-Gaussian), Laplace (RNADE-Laplace), sinh-arcsinh (RNADE-SAS), mixtures of Gaussians (RNADE-MoG) and mixtures of Laplace (RNADE-MOL).

3.3.3 Neural network alterations

As discussed by Bengio (2011), as a NADE (or an RNADE) with sigmoidal units progresses across the input dimensions $d \in \{1 \dots D\}$, its hidden units will tend to become more and more saturated, due to their input being a weighted sum of an

increasing number of inputs. Bengio proposed alleviating this effect by rescaling the hidden units' activation by a free factor ρ_d at each step, making the hidden unit values

$$h_{d,i} = \text{sigm}\left(\rho_d a_{d,i}\right). \tag{3.24}$$

A comparison of the statistical performance of RNADEs with or without these extra parameters is shown in Table 3.5 on page 57. Learning these extra rescaling parameters works slightly better. All of our experiments use them.

Previous work on neural networks has found that rectified linear units can work better than sigmoidal non-linearities (Nair and Hinton, 2010). The hidden values for rectified linear units are:

$$\boldsymbol{h}_d = \begin{cases} \rho_d \boldsymbol{a}_d & \text{if } \rho_d \boldsymbol{a}_d > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.25}$$

In preliminary experiments we found that these hidden units worked better than sigmoidal units in RNADE, and used them throughout (see results on natural image patches using several non-linearities on Table 3.4).

3.4 Experiments

We compared RNADE to mixtures of Gaussians (MoG) and factor analyzers (MFA), which are surprisingly strong baselines in some tasks (Tang et al., 2012; Zoran and Weiss, 2012). Given the known poor performance of discrete mixtures (Salakhutdinov and Murray, 2008; Larochelle and Murray, 2011), we limited our experiments to modeling continuous attributes. However it would be easy to include both discrete and continuous variables in a NADE-like architecture.

3.4.1 Low-dimensional data

We first considered three UCI datasets (Bache and Lichman, 2013), previously used to study the performance of other density estimators (Silva et al., 2011; Tang et al., 2012), namely: *red wine*, *white wine* and *parkinsons*. These datasets have low dimensionality (see Table 3.1), but the data has hard thresholds and non-linear dependencies that may make it difficult to fit mixtures of Gaussians or factor analyzers.

Following Tang et al. (2012), we eliminated discrete-valued attributes and an attribute from every pair with a Pearson correlation coefficient greater than 0.98.

Table 3.1: Dimensionality and size of the UCI dataset utilized in Section 3.4.1

	Red wine	White wine	Parkinsons
Dimensionality	11	11	15
Total number of datapoints	1599	4898	5875

Table 3.2: Average test-set log-likelihoods per datapoint for seven models on three UCI datasets. Performances not in bold can be shown to be significantly worse than at least one of the results in bold as per a paired t-test on the ten mean-likelihoods, with significance level 0.05.

Model	Red wine	White wine	Parkinsons
Gaussian	-13.18	-13.20	-10.85
MFA	-10.19	-10.73	-1.99
RNADE-FV	-12.29	-12.50	-8.87
RNADE-Gaussian	-11.99	-12.20	-3.47
RNADE-SAS	-9.86	-11.22	-3.07
RNADE-MoG	$\mathbf{-9.36}$	$\mathbf{-10.23}$	$\mathbf{-0.90}$
RNADE-MoL	$\mathbf{-9.46}$	-10.38	-2.63

Each dimension of the data was normalized by subtracting its training subset sample mean and dividing by its standard deviation. All results are reported on the normalized data.

As baselines we fitted full-covariance Gaussians and mixtures of factor analysers. To measure the performance of the different models, we calculated their log-likelihood on held-out test data. Because these datasets are small (see Table 3.1), we used 10-folds, with 90% of the data for training, and 10% for testing.

We chose the hyperparameter values for each model by doing per-fold cross-validation; using a ninth of the training data as validation data. Once the hyperparameter values had been chosen, we trained each model using all the training data (including the validation data) and measured its performance on the 10% of held-out testing data. In order to avoid overfitting, we stopped the training after reaching a training likelihood higher than the one obtained on the best validation-wise iteration of the corresponding validation run. Early stopping is crucial to avoid overfitting the RNADE models. It also improves the results of the MFAs, but to a lesser degree.

The MFA models were trained using the EM algorithm (Ghahramani and Hinton, 1996; Verbeek, 2005), the number of components and factors were cross-validated. The number of factors was chosen from even numbers from $2 \dots D$, where selecting D gives a mixture of Gaussians. The number of components was chosen among all even numbers (in order to halve the number of experiments) from $2 \dots 50$ (crossvalidation always selected fewer than 50 components).

The RNADE models were fitted using minibatch stochastic gradient descent (e.g. Duda et al., 2001), using minibatches of size 100, for 500 epochs, each epoch comprising 10 minibatches. For each experiment, the number of hidden units (50), the non-linear activation-function of the hidden units (ReLU), and the form of the conditionals were fixed. Three hyperparameters were cross-validated using grid-search: the number of components on each one-dimensional conditional (only applicable to the RNADE-MoG and RNADE-MoL models) was chosen from the set $\{2, 5, 10, 20\}$; the weight-decay (used only to regularize the input to hidden weights) from the set $\{2.0, 1.0, 0.1, 0.01, 0.001, 0\}$; and the learning rate from the set $\{0.1, 0.05, 0.025, 0.0125\}$. Learning-rates were decreased linearly to reach 0 after the last epoch.

The results are shown in Table 3.2. Autoregressive methods obtained statistical performances superior to mixture models on all datasets. An RNADE with mixture

of Gaussian conditionals was among the statistically significant group of best models on all datasets. As show in Figure 3.3 RNADE-SAS and RNADE-MoG models are able to capture hard thresholds and heterocedasticity.

Unfortunately we could not reproduce the data-folds used by previous work, however, our improvements are larger than those demonstrated by a deep mixture of factor analyzers over standard MFA (Tang et al., 2012).

3.4.2 Natural image patches

We also measured the ability of RNADE to model small patches of natural images. Following the recent work of Zoran and Weiss (2011), we use 8-by-8-pixel patches of monochrome natural images, obtained from the BSDS300 dataset (Martin et al., 2001) (Figure 3.4 gives examples).

Pixels in this dataset can take a finite number of brightness values ranging from 0 to 255. Modeling discretized data using a real-valued distribution can lead to arbitrarily high density values, by locating narrow high density spike on each of the possible discrete values. In order to avoid this 'cheating' solution, we added noise uniformly distributed between 0 and 1 to the value of each pixel. We then divided by 256, making each pixel take a value in the range $[0, 1]$.

In previous experiments, Zoran and Weiss (2011) subtracted the mean pixel value from each patch, reducing the dimensionality of the data by one: the value of any pixel could be perfectly predicted as minus the sum of all other pixel values. However, the original study still used a mixture of full-covariance 64-dimensional Gaussians. Such a model could obtain arbitrarily high model likelihoods, so unfortunately the likelihoods reported in previous work on this dataset (Zoran and Weiss, 2011; Tang et al., 2012) are difficult to interpret. In our preliminary experiment using RNADE, we observed that if we model the 64-dimensional data, the 64th pixel is always predicted by a very thin spike centered at its true value. The ability of RNADE to capture this spurious dependency is reassuring, but we wouldn't want our results to be dominated by it. Recent work by Zoran and Weiss (2012), projects the data on the leading 63 eigenvectors of each component, when measuring the model likelihood (Zoran, 2013). For comparison amongst a range of methods, we advocate simply discarding the 64th (bottom-right) pixel.

All of the results in this section were obtained by fitting the pixels in a raster-scan order.

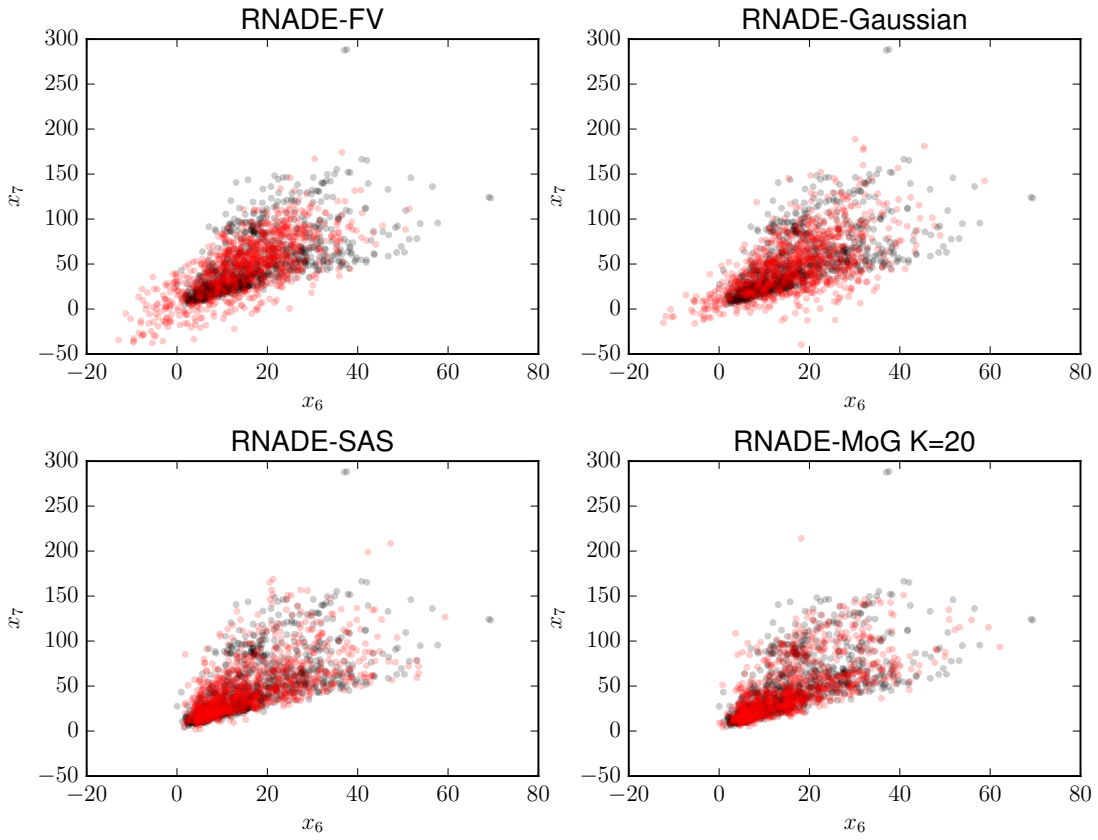

Figure 3.3: Scatter plot of dimensions x_7 vs x_6 of the *red wine* dataset. A thousand datapoints from the dataset are shown in black in all subfigures. As can be observed, this conditional distribution $p(x_7 \,|\, x_6)$ is heterocedastic, skewed and has hard thresholds. In red, a thousand samples from four RNADE models with different one-dimensional conditional forms are shown. **Top-left**: In red, one thousand samples from a RNADE-FV model. **Top-right**: In red, one thousand samples from a RNADE-Gaussian model. **Bottom-left**: In red, one thousand samples from a RNADE-SAS (sinh-arcsinh distribution) model. **Bottom-right**: In red, one thousand samples from a RNADE-MoG model with 20 components per one-dimensional conditional. The RNADE-SAS and RNADE-MoG models successfully capture all the characteristics of the data.

Experimental details follow. We trained our model using patches drawn randomly from 180 images in the training subset of BSDS300. A validation dataset containing 1,000 random patches from the remaining 20 images in the training subset were used for early-stopping when training RNADE. We measured the performance of each model by measuring their log-likelihood on one million patches drawn randomly from the test subset, which is composed of 100 images not present in the training subset. Given the larger scale of this dataset, hyperparameters of the RNADE and MoG models were chosen manually using the performance of preliminary runs on the validation data, rather than by an extensive search.

Unless specified otherwise, the RNADE models had $h = 512$ rectified-linear hidden units. Training was done by minibatch gradient descent, with 25 datapoints per minibatch, for a total of 1000 epochs, each comprising 1,000 minibatches. The learning-rate was scheduled to start at 0.001 (except for RNADE-SAS for which it was initialised to 0.00005), and linearly decreased to reach 0 after the last epoch. Gradient momentum with momentum factor 0.9 was used, but initiated at the beginning of the second epoch. A weight decay rate of 0.001 was applied to the input-to-hidden weight matrix only. We found that multiplying the gradient of the mean output parameters by the standard deviation improves results of the models with mixture outputs[4]. RNADE training was early stopped but didn't show signs of overfitting. Even larger models might perform better.

The MoG model was trained using minibatch EM, for 1,000 iterations. At each iteration 20,000 randomly sampled datapoints were used in an EM update. A step was taken from the previous mixture model towards the parameters resulting from the M-step: $\boldsymbol{\theta_t} = (1-\eta)\boldsymbol{\theta_{t-1}} + \eta\boldsymbol{\theta_{EM}}$, where the step size, η, was scheduled to start at 0.1 and linearly decreased to reach 0 after the last update. The training of the MoG was also early-stopped and also showed no signs of overfitting.

The results are shown in Table 3.3. We report the average log-density on a million image patches from the held out set of test images. The ranking of RNADE models is maintained when ordered by validation likelihood; the model with best test-likelihood would have been chosen using crossvalidation across all the RNADE models shown in the table. We also compared RNADE with a MoG trained by Zoran and Weiss (downloaded from Daniel Zoran's website) from which we removed the 64th row and column of each covariance matrix. There are

[4]Empirically, we found this to work better than regular gradients and also better than multiplying by the variances, which would provide a step with the right units.

two differences in the set-up of our experiments and those of Zoran and Weiss. First, we learnt the means of the MoG components, while Zoran and Weiss (2011) fixed them to zero. Second, we held-out 20 images from the training set to do early-stopping and hyperparameter optimisation, while they used the 200 images for training.

As we expected, the RNADE-FV model with fixed conditional variances obtained very low statistical performance. Adding an output parameter per dimension to have variable standard deviations made our models competitive with MoG with 100 full-covariance components. However, in order to obtain results superior to the mixture of Gaussians model trained by Zoran and Weiss, we had to use richer conditional distributions: one-dimensional mixtures of Gaussians (RNADE-MoG). The best RNADE model obtained on average, 0.9 nats per patch higher log-density than Zoran and Weiss's MoG.

In Figure 3.4 we show one hundred examples from the test set, one hundred examples from Zoran and Weiss' mixture of Gaussians, and a hundred samples from our best RNADE-MoG model. Similar patterns can be observed in the three cases: uniform patches, edges, and locally smooth noisy patches.

The marginal for the brightness of a single pixel in natural image patches is heavy tailed, closer to a Laplace distribution than a Gaussian. An RNADE with Laplace conditionals (RNADE-Laplace) obtained higher test-likelihood than an RNADE with Gaussian conditionals (RNADE-Gaussian). However, using mixture models as conditionals, likelihoods are higher for Gaussian components. As can be seen in Figure 3.6, both the RNADE-MoG and RNADE-MoL can very accurately match the empirical distribution over the first dimension. The RNADE-MoG must fit predictions of the first pixel, $p(x_1)$, with several Gaussians of different widths, that coincidentally have zero mean. However, later pixels were predicted better with Gaussian outputs (Figure 3.5); the mixture of Laplace model is not suitable for predicting with large contexts. In Figure 3.7 we compare two conditional distributions from an RNADE-MoG and an RNADE-MoL which point to a plausible explanation: mixing Laplace components with different means results in a distribution with several density "peaks", while the mixtures of Gaussians can still produce smooth unimodal conditionals.

Table 3.3: Average per-example log-likelihood of several mixture of Gaussian and RNADE models on 8-by-8 pixel patches of natural images. These results are measured in nats and were calculated using one million patches. Standard errors due to the finite test sample size are lower than 0.1 nats in every case. K gives the number of one-dimensional components for each conditional in RNADE, and the number of full-covariance components for MoG.

Model	Test log-likelihood
MoG $K=200$ (Zoran and Weiss, 2012)[a]	152.8
MoG $K=100$	144.7
MoG $K=200$	150.4
MoG $K=300$	150.4
RNADE-FV	100.3
RNADE-Gaussian	143.9
RNADE-Laplace	145.9
RNADE-SAS[b]	148.5
RNADE-MoG $K=2$	149.5
RNADE-MoG $K=2$ $h=1024$	150.3
RNADE-MoG $K=5$	152.4
RNADE-MoG $K=5$ $h=1024$	152.7
RNADE-MoG $K=10$	153.5
RNADE-MoG $K=10$ $h=1024$	**153.7**
RNADE-MoL $K=2$	149.3
RNADE-MoL $K=2$ $h=1024$	150.1
RNADE-MoL $K=5$	151.5
RNADE-MoL $K=5$ $h=1024$	151.4
RNADE-MoL $K=10$	152.3
RNADE-MoL $K=10$ $h=1024$	152.5

[a] This model was trained using the full 200 images in the BSDS training dataset, the rest of models were trained using 180, reserving 20 for hyperparameter crossvalidation and early-stopping.

[b] Training an RNADE with sinh-arcsinh conditionals required the use of a starting learning rate 20 times smaller to avoid divergence during training. For this reason, this model was trained for 2000 epochs.

Table 3.4: Comparison of test-log-likelihoods for three RNADE-MOG models with 5 Gaussian components per conditional and 512 hidden units trained using different non-linearities for the hidden layer. Standard-errors due to the finite size of the test dataset are smaller than 0.1 in the three cases.

Nonlinearity	Test log-likelihood
Rectified linear units (ReLU)	152.4
Sigmoid	149.6
Hyperbolic tangent (tanh)	148.9

Table 3.5: Statistics for the test-log-likelihood of RNADE-Gaussian models trained with or without the extra parameters for the rescaling of hidden unit activations described in Section 3.3.3.

Order	Runs	Min	Max	Median	Mean	St. dev.
No rescaling	15	142.3	142.4	142.4	142.4	0.04
Rescaling	15	143.9	144.1	144.0	144.0	0.06

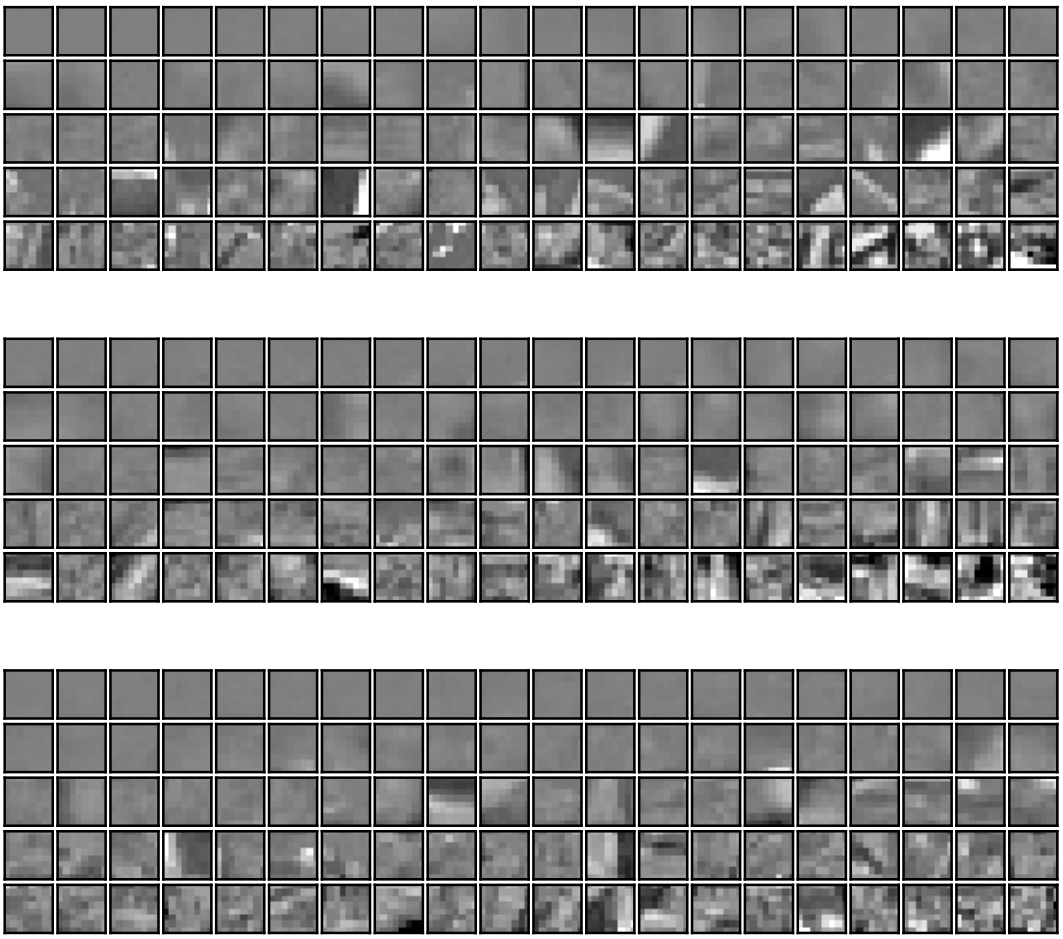

Figure 3.4: **Top:** 100 8x8 patches from the BSDS test set. **Center:** 100 samples from a mixture of Gaussians with 200 full-covariance components. **Bottom:** 100 samples from an RNADE with 1024 hidden units and 10 Gaussian components per conditional. All data and samples were drawn randomly and sorted by their density under the RNADE.

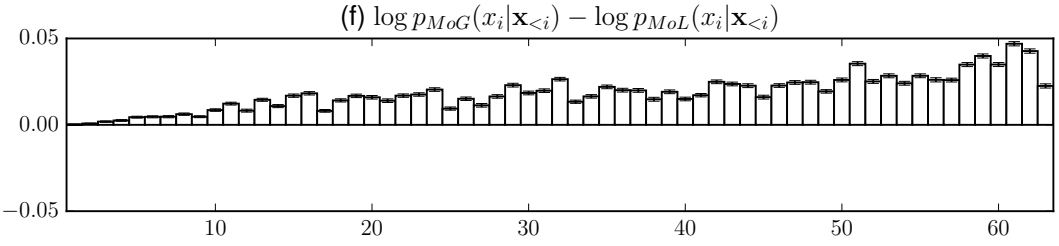

Figure 3.5: Average difference in log-density between a RNADE-MoG and an RNADE-MoL for each pixel. One-standard-error bars are shown.

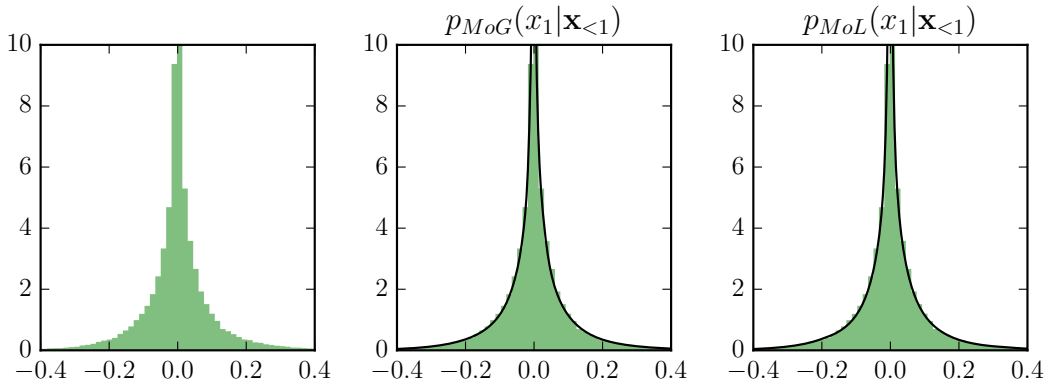

Figure 3.6: **Left:** Histogram of pixel intensities in the BSDS training dataset after mean brightness removal. **Centre:** Density function for $p(x_1)$ under an RNADE with mixture of Gaussian conditionals superimposed on the empirical histogram. **Right:** Density function for $p(x_1)$ under an RNADE with mixture of Gaussian conditionals superimposed on the empirical histogram.

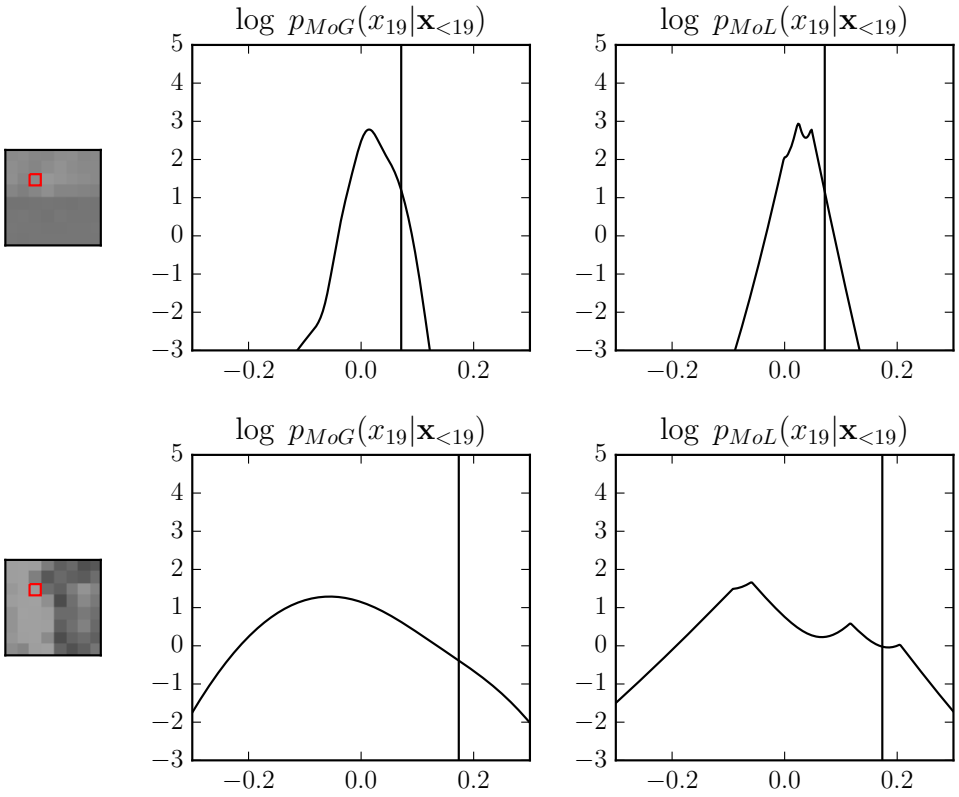

Figure 3.7: One-dimensional conditionals for the 19th pixel (shown in red on the left) conditioned on the previous 18 pixels for two natural image patches. **Left:** The patch. **Centre:** Conditional under an RNADE-MoG **Right:** Conditional under an RNADE-MoL. RNADE-MoL results in conditionals with peaks of log-density.

3.4.3 Speech acoustics

We also measured the ability of RNADE to model small patches of speech spectrograms, extracted from the TIMIT dataset (Garofolo et al., 1993). The patches contained 11 frames of 20 filter-banks plus energy; totalling 231 dimensions per datapoint. These filter-bank encoding is common in speech-recognition, and better for visualization than the more frequently used cepstral coefficient features. A good generative model of speech acoustics could be used, for example, in denoising, or speech detection tasks.

We fitted the models using the standard TIMIT training subset, which includes recordings from 605 speakers of American English. We compare RNADE with a mixture of Gaussians by measuring their log-likelihood on the complete TIMIT core-test dataset: a held-out set of 25 speakers.

The RNADE models have $h = 512$ rectified-linear hidden units and a mixture of 20 one-dimensional Gaussian components per output. Given the large scale of this dataset, hyperparameter choices were again made manually using validation data. The same minibatch training procedures for RNADE and mixture of Gaussians were used as for natural image patches.

Training was done using minibatch gradient descent, with 25 datapoints per minibatch, for a total of 200 epochs, each comprising 1,000 minibatches. The learning-rate was scheduled to start at 0.001 and linearly decreased to reach 0 after the last epoch. Gradient momentum with momentum factor 0.9 was used, but initiated at the beginning of the second epoch. A weight decay rate of 0.001 was applied to the input-to-hidden weight matrix only. Again, we found that multiplying the gradient of the mean output parameters by the standard deviation improved results. RNADE training was early stopped but didn't show signs of overfitting.

The MoG model was trained using minibatch EM, for 1,000 iterations. At each iteration 20,000 randomly sampled datapoints were used to calculate an EM step, updating the current mixture model by interpolating the current parameters and the new parameters resulting from the M step $\boldsymbol{\theta_t} = (1-\eta)\boldsymbol{\theta_{t-1}} + \eta\boldsymbol{\theta_{EM}}$ for 1,000 updates, where the step size (η) was scheduled to start at 0.1 and linearly decreased to reach 0 after the last update. The training of the MoG was also early-stopped and also showed no signs of overfitting.

The results are shown in Table 3.6. The best RNADE (which would have

Table 3.6: Log-likelihood of several MoG and RNADE models on the core-test set of TIMIT measured in nats. Standard errors due to the finite test sample size are lower than 0.4 nats in every case. RNADE obtained a higher (better) log-likelihood.

Model	Test LogL
MoG $N=50$	110.4
MoG $N=100$	112.0
MoG $N=200$	112.5
MoG $N=300$	112.5
RNADE-Gaussian	110.6
RNADE-Laplace	108.6
RNADE-SAS	119.2
RNADE-MoG $K=2$	121.1
RNADE-MoG $K=5$	124.3
RNADE-MoG $K=10$	**127.8**
RNADE-MoL $K=2$	116.3
RNADE-MoL $K=5$	120.5
RNADE-MoL $K=10$	123.3

been selected based on validation results) obtained, on average, 15 nats more per test example than a mixture of Gaussians. In Figure 3.8 examples from the test set, and samples from the MoG and RNADE-MoG models are shown. In contrast with the log-likelihood measure, there are no marked differences between the samples from each model. Both set of samples look like blurred spectrograms, but RNADE seems to capture sharper formant structures (peaks of energy at the lower frequency bands characteristic of vowel sounds).

3.5 Sensitivity to the ordering of dimensions

RNADEs with different orders of the variables have different inductive biases and can have different statistical performances for a given dataset. As an illustration, modelling the dataset shown in Figure 3.1(c) using an RNADE with Gaussian conditionals can have very different statistical performance for the two orders

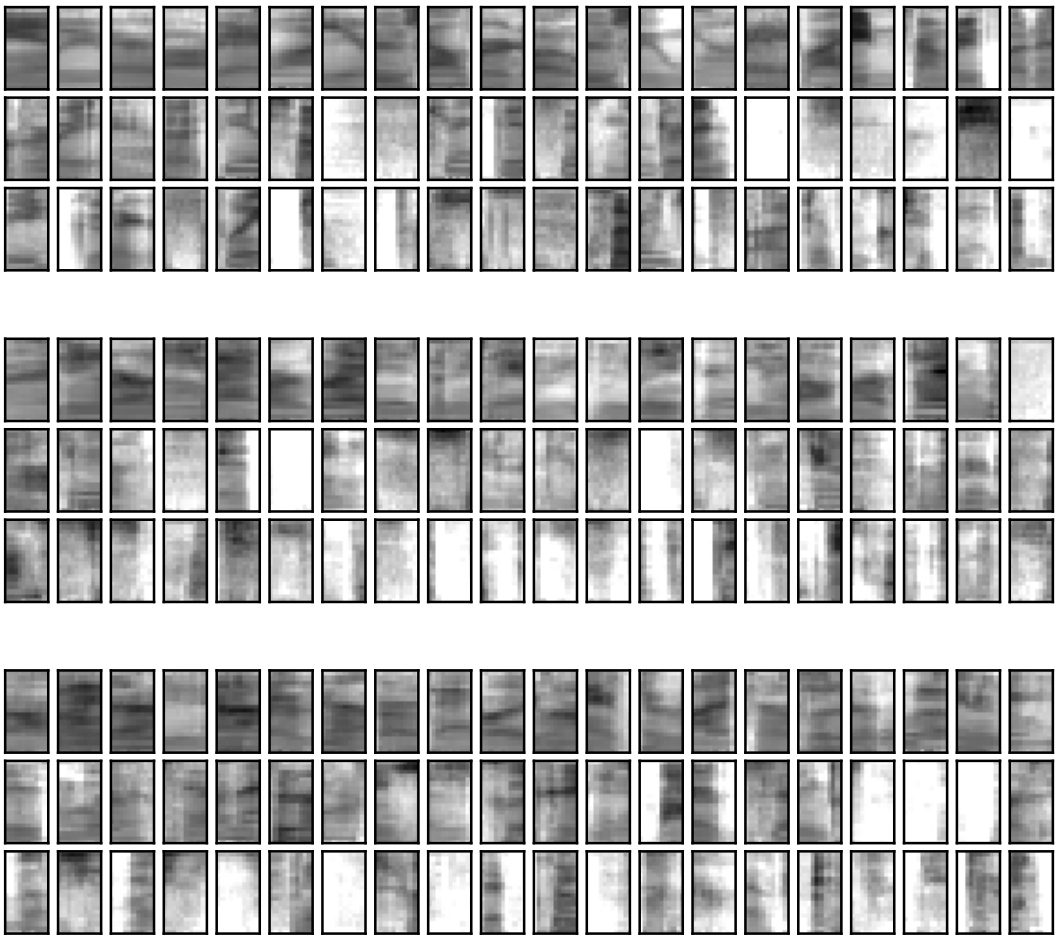

Figure 3.8: **Top:** 60 datapoints from the TIMIT core-test set. **Center:** 60 samples from a MoG model with 200 components. **Bottom:** 60 samples from an RNADE with 512 hidden units and 10 Gaussian output components per dimension. For each datapoint displayed, time is shown on the horizontal axis, the bottom row displays the energy feature, while the others display the Mel filter bank features (in ascending frequency order from the bottom). All data and samples were drawn randomly and sorted by density under the RNADE model.

possible. We would expect much higher performance using the order $\{x_2, x_1\}$ as both $p(x_2)$ and $p(x_1 \,|\, x_2)$ are unimodal distributions. However, the order $\{x_1, x_2\}$ will require modelling the multimodal conditional distribution $p(x_2 \,|\, x_1)$.

Alas, finding the best ordering of the dimensions is a combinatorial optimization problem. Checking the performance under a particular ordering requires training an RNADE model, which makes exhaustive search intractable for problems with more than a few dimensions. Techniques that allow learning the connectivity

Table 3.7: Statistics for the test-log-likelihood (in nats) of 15 RNADE-Gaussian models trained using random orders of the dimensions or raster-scan order. Random orders are superior.

Order	Runs	Min	Max	Median	Mean	St. dev.
Scan-order	15	143.9	144.1	144.0	144.0	0.06
Random orders	15	144.8	145.1	144.9	144.9	0.07

structure of a Bayes net (Chow and Liu, 1968; Buntine, 1991) can be used to find a partial ordering of the dimensions. Also, given domain knowledge about the causal relationship of the variables modelled, a heuristic ordering of the variables, in which causes precede effects can be expected to be easier to model (Guyon et al., 2007).

All experiments presented so far were done on a single ordering of the dimensions for each dataset. For the UCI datasets we utilized the ordering in the UCI data files; for natural images raster-scan ordering, i.e. top-down and left-to-right; for speech acoustics the features were ordered by time, and within each time-frame from low to high frequency, followed by the energy feature.

To test the importance of the order for natural image patches, we trained 15 RNADE-Gaussian models with different orders chosen at random and 15 RNADE-Gaussian using scan-order (top to bottom, left to right) with different seeds for the random number generator. The results are shown in Table 3.7. All 15 picked-at-random orders obtained a test-log-likelihood between 144.8 and 145.1 nats, while the scan-order models obtained log-likelihoods between 143.9 and 144.0. The mean of the distribution test-likelihoods for orders picked at random was judged as greater than that of raster-scan order RNADEs by a t-test at significance level $\alpha = 0.01$.

Note that these results are maintained if we use the likelihoods for the validation dataset, which would allow us to claim the ability to choose a random order for our experiments without overfitting to the test data. We show test-log-likelihood for easier comparison to the rest of results shown in other tables.

However, the difference in performance among random orders was comparable to that among runs that use the same ordering but different random-number-generator seeds. We cannot exclude the possibility of even better orderings, but

finding them would require combinatorial optimization techniques.

We would also like to know whether the advantage of using a random order is maintained when RNADEs with more capable one-dimensional conditional forms are used. To test this, we re-trained the best architecture in Table 3.3 on page 56; i.e. an RNADE-MoG with 10 components per conditional and 1024 hidden units using the random-ordering that obtained highest validation likelihood for the experiments in Table 3.7. We obtained a test-log-likelihood of 153.7 nats, which coincides with that of the raster-scan ordering (the validation likelihood was also the same up to the first decimal). This seems to indicate that the advantage of using random orderings for this dataset when using a simple Gaussian conditional does not transfer to more complex conditional forms.

The advantage of random orders for a Gaussian conditional is surprising, but its absence in the case of multimodal conditionals points to a possible explanation. Knowing the value of pixels from different regions of the patch can detect the presence or absence of an edge early on, while scan-order modelling has to hedge for its possible presence at every pixel by increasing the variance of the conditionals. A mixture form for the conditionals would be less affected by this because a component with small weight and high variance can be used to hedge for the presence of edges.

3.6 Discussion

Mixture Density Networks (MDNs) (Bishop, 1994) are a flexible conditional model of probability densities, that can capture skewed, heavy-tailed, and multi-modal distributions. In principle, MDNs can be applied to multi-dimensional data. However, the number of parameters that the network has to output grows quadratically with the number of targets, unless the targets are assumed independent. RNADE exploits an autoregressive framework to apply practical, one-dimensional MDNs to unsupervised density estimation.

Modelling real-valued data with a series of one-dimensional conditionals is more challenging than modelling binary data. Any one dimensional distribution over a binary variable is given by a Bernoulli distribution. However, when modelling real-valued data, a specific parametric form for each conditional must be chosen. This choice, and the limited ability of the neural network to map its inputs to the parameters of the conditionals, introduces an inductive bias. A sufficiently

large mixture of Gaussians can represent any density, and a one-hidden-layer neural network with enough hidden units can approximate any function to an arbitrary precision. However, these theoretical assurances offer little comfort in the real-world scenario of limited data and computational resources. Other one-dimensional forms than the ones analyzed in this chapter may aid RNADE to generalize better to different context sizes and across a range of applications.

One of the main drawbacks of RNADE, and of neural networks in general, is the need to decide the value of several training hyperparameters. The gradient descent learning rate can be adjusted automatically using, for example, the techniques developed by Schaul et al. (2013). Also, methods for choosing hyperparameters more efficiently than grid search have been recently developed (Bergstra and Bengio, 2012; Snoek et al., 2012). These, and several other recent improvements in the neural network field, like dropouts (Srivastava et al., 2014), should be directly applicable to RNADE, and possibly obtain even better performance than shown in this work. RNADE makes it relatively straight-forward to translate advances in the neural-network field into better density estimators, or at least into new estimators with different inductive biases.

In summary, in this chapter we have presented RNADE, a novel 'black-box' density estimator. Both likelihood computation time and the number of parameters scale linearly with the dataset dimensionality. Generalization across a range of tasks, representing arbitrary feature vectors, image patches, and auditory spectrograms is excellent.

Chapter 4

A deep and tractable density estimator

This chapter is an extended version of the article "A deep and tractable density estimator" (Uria et al., 2014) published in ICML 2014.

In this chapter we introduce an efficient procedure to simultaneously train a NADE model for each possible ordering of the variables, by sharing parameters across all these models (Section 4.2). We can thus use the most convenient model for each inference task at hand, and ensembles of such models with different orderings are immediately available (Section 4.3). Moreover, unlike the original NADE, our training procedure scales to deep models. Empirically, ensembles of deep NADE models obtain state of the art density estimation performance (Section 4.5).

4.1 Introduction

NADE and RNADE, have been shown to be state-of-the-art joint density models for a variety of real-world datasets (Larochelle and Murray, 2011) (also previous Chapter), as measured by their predictive likelihood. These models predict each variable sequentially in an arbitrary order, fixed at training time. Variables at the beginning of the order can be set to observed values, i.e., conditioned on. Variables at the end of the ordering are not required to make predictions; marginalizing these variables requires simply ignoring them, see Equation (2.73). However, marginalizing over and conditioning on any arbitrary subsets of variables will not be easy in general.

Another disadvantage of NADE compared to other neural network models is

that an efficient deep formulation (e.g. Bengio, 2009) is not available. While extending NADE's definition to multiple hidden layers is trivial, we simply introduce regular feed-forward layers between the computation of the hidden units in (3.20) and outputs in (3.17, 3.18, and 3.19), we lack an efficient recursive expression like (3.22 and 3.23) for the added layers. Thus, when NADE has more than one hidden layer, each additional hidden layer must be computed separately for each input dimension, yielding a complexity cubic on the size of the layers $O(DH^2L)$, where L represents the number of layers. This scaling seemingly made a deep NADE impractical, except for datasets of low dimensionality.

In this chapter, we present a procedure for training a factorial number of NADE (or RNADE) models simultaneously; one for each possible ordering of the variables. The parameters of these models are shared, and we optimize the mean cost over all orderings using a stochastic gradient technique. After fitting the shared parameters, we can extract, in constant time, the NADE model with the variable ordering that is most convenient for any given inference task. While the different NADE models might not be consistent in their probability estimates, this property is actually something we can leverage to our advantage, by generating ensembles of NADE models "on the fly" (i.e., without explicitly training any such ensemble) which are even better estimators than any single NADE. In addition, our procedure is able to train a deep version of NADE, incurring an extra computational expense only linear in the number of layers.

4.2 Training a factorial number of NADEs

Looking at the simplicity of inference in Equation (2.73), a naive approach that could exploit this property for any inference task would be to train as many NADE models as there are possible orderings of the input variables. This approach, requiring $O(D!)$ time and memory, is not viable. However, we show here that through some careful parameter tying between models, we can derive an efficient stochastic procedure for training all models, minimizing the mean of their negative log-likelihood objectives.

Consider for now a parameter tying strategy that simply uses the same weight matrices and bias parameters across all NADE models (we will refine this proposal later). We will now write $p(\boldsymbol{x} \,|\, \boldsymbol{\theta}, o)$ as the joint distribution of the NADE model that uses ordering o and $p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d)$ as its associated conditionals,

which are computed as specified in Section 2.8.3 (Section 3.3 for RNADE), or its straightforward extension in the deep network case. Thus we explicitly treat the ordering o as a random variable. Notice that the d^{th} conditional only depends on the first d elements of the ordering, and is thus exactly the same across NADE models sharing their first d elements in o. During training we will attempt to minimise $\mathcal{J}_{OA}$ (for order-agnostic loss), the expected (over variable orderings) negative log-likelihood of the model for the training data:

$$\mathcal{J}_{OA}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{o \in D!} -\log p(\boldsymbol{X} \,|\, \boldsymbol{\theta}, o) \tag{4.1}$$

$$\propto \mathop{\mathbb{E}}_{o \in D!} \mathop{\mathbb{E}}_{\boldsymbol{x^{(n)}} \in \boldsymbol{X}} -\log p(\boldsymbol{x^{(n)}} \,|\, \boldsymbol{\theta}, o), \tag{4.2}$$

where $D!$ is the set of all orderings (i.e. permutations of D elements). We note this objective does not correspond to a mixture model, in which case the expectation over orderings would be inside the log operation.

Using NADE's autoregressive expression for the density of a datapoint, (4.2) can be rewritten as:

$$\mathcal{J}_{OA}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{o \in D!} \mathop{\mathbb{E}}_{\boldsymbol{x^{(n)}} \in \boldsymbol{X}} \sum_{d=1}^{D} -\log p(x^{(n)}_{o_d} \,|\, \boldsymbol{x}^{(n)}_{o_{<d}}, \boldsymbol{\theta}, o)\,, \tag{4.3}$$

where d indexes the elements in the order, o, of the dimensions. By moving the expectation over orders inside the sum over the elements of the order, the order can be split in three parts: $o_{<d}$ standing for the index of the $d-1$ first dimensions in the ordering; o_d the index of the d-th dimension in the ordering, and $o_{>d}$ standing for the indices of the remaining dimensions in the ordering. Therefore, the loss function can be rewritten as:

$$\mathcal{J}_{OA}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{x^{(n)}} \in \boldsymbol{X}} \sum_{d=1}^{D} \mathop{\mathbb{E}}_{o_{<d}} \mathop{\mathbb{E}}_{o_d} \mathop{\mathbb{E}}_{o_{>d}} -\log p(x^{(n)}_{o_d} \,|\, \boldsymbol{x}^{(n)}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d)\,. \tag{4.4}$$

The value of each term does not depend on $o_{>d}$. Therefore, it can be simplified as:

$$\mathcal{J}_{OA}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{x^{(n)}} \in \boldsymbol{X}} \sum_{d=1}^{D} \mathop{\mathbb{E}}_{o_{<d}} \mathop{\mathbb{E}}_{o_d} -\log p(x^{(n)}_{o_d} \,|\, \boldsymbol{x}^{(n)}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d)\,. \tag{4.5}$$

In practice, this loss function (4.5) will have a very high number of terms and will have to be approximated by sampling $\boldsymbol{x}^{(n)}$, d, and $o_{<d}$ uniformly. The innermost expectation over values of o_d can be calculated cheaply for a NADE given that

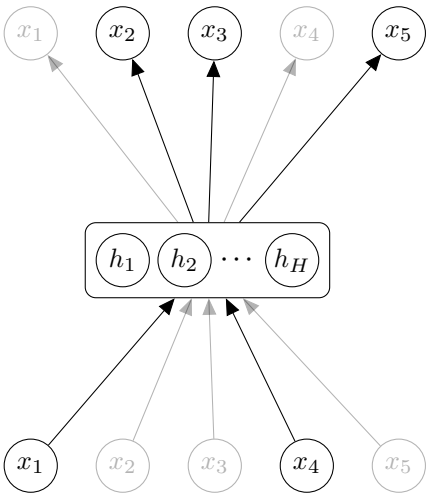

Figure 4.1: Graphical representation of a feed-forward neural network that estimates $\mathcal{J}_{OA}$ for an orderless-NADE that models a 5-dimensional variable. In this particular instance of the estimator $d = 3$, and $o_{<d} = \{x_1, x_4\}$. The estimation calculated is: $\widehat{\mathcal{J}_{OA}} = -\frac{5}{3}(\log p(x_2 \,|\, x_1, x_4) + \log p(x_3 \,|\, x_1, x_4) + \log p(x_5 \,|\, x_1, x_4))$.

the hidden unit states $\boldsymbol{h}_d$ are shared for all possible o_d. Therefore, assuming all orderings are equally probable, we will estimate $\mathcal{J}_{OA}(\boldsymbol{\theta})$ by:

$$\widehat{\mathcal{J}_{OA}}(\boldsymbol{\theta}) = \frac{D}{D-d+1} \sum_{o_d} -\log p(x^{(n)}_{o_d} \,|\, \boldsymbol{x}^{(n)}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d)\,, \tag{4.6}$$

which provides an unbiased estimator of (4.2). Thus training can be done by descent on the stochastic gradient of $\widehat{\mathcal{J}_{OA}}(\boldsymbol{\theta})$. This method corresponds to a very simple procedure where the number of inputs present is sampled uniformly from $0 \ldots D-1$, a random subset of dimensions of that size is picked as inputs, and the rest of dimensions are predicted as the next dimension in the ordering and rescaled by $\frac{D}{D-d+1}$ to account for the reduced number of outputs. An implementation of this order-agnostic density estimator corresponds to a single feedforward pass through an artificial neural network with D inputs and D outputs, or a mixture density network in the case of real-valued data (see Figure 4.1).

The end result is a stochastic training update costing $O(DH + H^2 L)$, as in regular multilayer neural networks. At test time, sampling and density calculation of a NADE with a single hidden layer remains $O(DH)$. We unfortunately cannot avoid a complexity of $O(DH^2 L)$ in NADEs with several hidden layers, and perform D passes through the neural network to obtain all D conditionals for some given ordering. However, this is still tractable (low-degree polynomial complexity), unlike, for example, computing probabilities in a restricted Boltzmann machine

or a deep belief network.

4.2.1 Improved parameter sharing using input masks

While the parameter tying proposed so far is simple, in practice it leads to poor statistical performance. One issue is that the values of the hidden units, computed using (3.20), are the same when a dimension is in $\boldsymbol{x}_{o_{>d}}$ (a value not present in the inputs) and when the value of that dimension is zero and conditioned on. When training just one NADE with a fixed o, each output unit knows which inputs feed into it, but in the multiple ordering case that information is lost when the input is zero.

In order to make this distinction possible, we augment the parameter sharing scheme by appending to the inputs a binary mask vector $\boldsymbol{m}_{o_{<d}} \in \{0,1\}^D$ indicating which dimensions are present in the input. That is, the i-th element of $\boldsymbol{m}_{o_{<d}}$ is 1 if $i \in o_{<d}$ and 0 otherwise. A graphical representation of this technique is shown in Figure 4.2. One interpretation of this modification is that the bias vector $\boldsymbol{c}$ of the first hidden layer is now dependent on the ordering o and the value of d, thus slightly relaxing the strength of parameter sharing between the NADE models. We have found in practice that this adjustment is crucial to obtain good density estimation performance. Some results showing the difference in statistical performance with and without training masks can be seen in Table 4.2 as part of our experimental analysis (see Section 4.5 for details).

Therefore, we will substitute the expression for the hidden unit states (3.20) with:

$$\boldsymbol{h}_d = \text{sigm}\left(\boldsymbol{W}_{\cdot,o_{<d}}\boldsymbol{x}_{o_{<d}} + \boldsymbol{U}\boldsymbol{m}_{o_{<d}} + \boldsymbol{c}\right), \tag{4.7}$$

where $U \in \mathbb{R}^{H \times D}$. This modification still admits a recursive expression of the first hidden layer activations:

$$\boldsymbol{a}_1 = \boldsymbol{c} \tag{4.8}$$

$$\boldsymbol{a}_{d+1} = \boldsymbol{a}_d + x_{o_d}\boldsymbol{W}_{\cdot,o_d} + \boldsymbol{U}_{\cdot,o_d}, \tag{4.9}$$

which allows us to maintain the mild run-time complexity of one-hidden-layer NADEs.

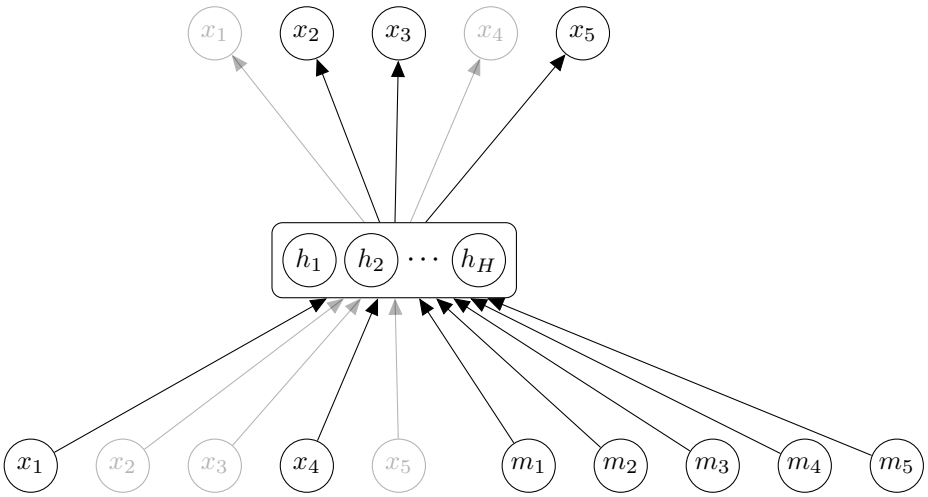

Figure 4.2: Graphical representation of a feed-forward neural network that estimates $\mathcal{J}_{OA}$ for an orderless-NADE with input masks. This NADE models a 5-dimensional variable. In this particular instance of the estimator sampling chose values $d = 3$, and $o_{<d} = \{x_1, x_4\}$. The mask, which depends deterministically on d and o, takes value $\boldsymbol{m} = (1, 0, 0, 1, 0)$. The estimate calculated is:
$\widehat{\mathcal{J}_{OA}} = -\frac{5}{3}\left(\log p(x_2 \,|\, x_1, x_4, \boldsymbol{m}) + \log p(x_3 \,|\, x_1, x_4, \boldsymbol{m}) + \log p(x_5 \,|\, x_1, x_4, \boldsymbol{m})\right)$.

4.3 On the fly generation of NADE ensembles

Our order-agnostic training procedure can be thought of as producing a set of parameters that can be used by a factorial number of NADEs, one per ordering of the input variables. These different NADEs will not, in general, agree on the probability of a given datapoint. While this disagreement might look unappealing at first, we can actually use this source of variability to our advantage, and obtain better estimates than is possible with a set of consistent models.

A NADE with a given input ordering corresponds to a different hypothesis space than other NADEs with different ordering. In other words, each NADE with a different ordering is a model in its own right, with slightly different inductive bias, despite the parameter sharing.

A reliable approach to improve on some given estimator is to instead construct an ensemble of multiple, strong but different estimators, e.g. with bagging (Ormoneit and Tresp, 1995) or stacking (Smyth and Wolpert, 1999). Our training procedure suggest a straightforward way of generating ensembles of NADE models: generate a set of uniformly distributed orderings $\{o^{(k)}\}_{k=1}^{K}$ over the input variables and use the average probability $\frac{1}{K}\sum_{k=1}^{K} p(\boldsymbol{x}|\theta, o^{(k)})$ as our estimator.

Ensemble averaging increases the computational cost of density estimation linearly with the size of the ensemble, while the complexity of sampling doesn't change (we pick one of the K orderings, $o^{(k)}$, at random from the ensemble and sample from the corresponding NADE). Importantly, the computational cost of training remains the same, unlike ensemble methods such as bagging. Moreover, an adequate number of components can be chosen after training, and can even be adapted to the available computational budget on the fly.

4.4 Related work

As mentioned previously, autoregressive density/distribution estimation has been explored before by others. For the binary data case, Frey (1998) considered the use of logistic regression conditional models, while Bengio and Bengio (2000) proposed a single layer neural network architecture, with a parameter sharing scheme different from the one in the NADE model (Larochelle and Murray, 2011). In all these cases however, a single (usually random) input ordering was chosen and maintained during training.

Frey (1998) already used ensembles of autoregressive models. He noted the difficulty of finding a good ordering and used a mixture of models; each trained using a different ordering of the variables. He also trained ensembles of models using different initial parameters and justified this as approximate Bayesian inference over the parameters.

Gregor and LeCun (2011) proposed training a variant of the NADE architecture under stochastically generated random orderings. Like us, they observed much worse performance than when choosing a single variable ordering, which motivates our proposed parameter sharing scheme relying on input masks. Gregor and LeCun generated a single ordering for each training update, and conditioned on contexts of all possible sizes to compute the log-probability of an example and its gradients. Our stochastic approach uses only a single conditioning configuration for each update, but computes the average log-probability for the next dimension under all possible remaining orderings. This change allowed us to generalize NADE to deep architectures with an acceptable computational cost.

Goodfellow et al. (2013) introduced a procedure to train deep Boltzmann machines by maximizing a variational approximation of their generalised pseudo-likelihood. This results in a training procedure similar to the one presented in

this work, where a subset of the dimension is predicted given the value of the rest.

Our algorithm also bears similarity with denoising autoencoders (Vincent et al., 2008) trained using so-called "masking noise". There are two crucial differences however. The first is that our procedure corresponds to training on the average reconstruction of only the inputs that are missing from the input layer. The second is that, unlike denoising autoencoders, the NADE models that we train can be used as tractable density estimators.

Our training technique also resembles *dropouts* (Srivastava et al., 2014). Dropouts is a regularization technique used for training neural networks with many hidden units. As the number of hidden units in a neural network is increased, its bias is reduced, and parameter fitting becomes more prone to overfitting to the training dataset. To avoid overfitting, the dropouts technique sets to zero with a certain probability the output of hidden and input units. At test time, no outputs are set to zero, and weights are rescaled to account for the higher expected number of inputs to each unit.

Both our training procedure and dropouts set some of the inputs to zero. However, the purpose and the details of each technique are different. The purpose of dropouts is regularization, while the purpose of our procedure is to obtain an unbiased estimator of the order-agnostic loss for a NADE. In our training procedure, the number of inputs to be set to zero is sampled uniformly from 1 to D, while in dropouts each input is dropped independently with a certain probability. Note that under dropouts it is almost impossible to drop all inputs, while in our procedure this will happen as often as any other number of them. Also, our procedure does not drop any hidden unit outputs.

Dropouts can be combined with our training procedure to avoid overfitting of NADE and RNADE models with many hidden units.

4.5 Experimental results

We performed experiments on several binary and real-valued datasets to asses the performance of NADEs trained using our order-agnostic procedure. We report the average test log-likelihood of each model, that is, the average log-density of datapoints in a held-out test set. In the case of NADEs trained in an order-agnostic way, we need to choose an ordering of the variables so that one may calculate the density of the test datapoints. We report the average test log-likelihoods over ten

Table 4.1: Average test-set log-likelihood per datapoint (in nats) of different models on eight binary datasets from the UCI repository. Baseline results were taken from Larochelle and Murray (2011).

Model	Adult	Connect4	DNA	Mushrooms	NIPS-0-12	Ocr-letters	RCV1	Web
MoBernoullis	−20.44	−23.41	−98.19	−14.46	−290.02	−40.56	−47.59	−30.16
RBM	−16.26	−22.66	−96.74	−15.15	−277.37	−43.05	−48.88	−29.38
FVSBN	**−13.17**	−12.39	**−83.64**	−10.27	−276.88	−39.30	−49.84	−29.35
NADE (fixed order)	−13.19	**−11.99**	−84.81	**−9.81**	**−273.08**	**−27.22**	**−46.66**	−28.39
NADE 1hl	−13.51	−13.04	−84.28	−10.06	−275.20	−29.05	−46.79	−28.30
NADE 2hl	−13.53	−12.99	−84.30	−10.05	−274.69	−28.92	−46.71	**−28.28**
NADE 3hl	−13.54	−13.08	−84.37	−10.10	−274.86	−28.89	−46.76	−28.29
EoNADE 1hl (2 ord)	−13.35	−12.81	−83.52	−9.88	−274.12	−28.36	−46.50	−28.11
EoNADE 1hl (16 ord)	−13.19	−12.58	**−82.31**	**−9.68**	**−272.38**	−27.31	**−46.12**	**−27.87**

different orderings chosen at random. Note that this is different from an ensemble, where the probabilities are averaged before calculating its logarithm. To reduce clutter, we have not reported the standard deviation across orderings. In all cases, this standard deviation has magnitude smaller than the log-likelihood's standard error due to the finite size of our test sets. These standard errors are also small enough not to alter the ranking of the different models. In the case of ensembles of NADEs the standard deviation due to different sets of orderings is, as expected, even smaller. Every results table is partitioned in three parts by horizontal lines, the top part contains baselines, the middle part results obtained using the orderless training procedure, and the bottom part the results of ensembles of NADEs. In every table the log-likelihood of the best single model, and the log-likelihood of the best ensemble are shown in bold.

Training configuration details common to all datasets (except where specified later on) follow. We trained all order-agnostic NADEs and RNADEs using minibatch stochastic gradient descent on $\mathcal{J}_{OA}$, (4.5). The initial learning rate, which was chosen independently for each dataset, was reduced linearly to reach zero after the last iteration. For the purpose of consistency, we used rectified linear units (Nair and Hinton, 2010) in all experiments. We found that this type of unit allow us to use higher learning rates and made training converge faster. We used Nesterov's accelerated gradient (Sutskever, 2013) with momentum value 0.9. No weight decay was applied. To avoid overfitting, we early-stopped training by estimating the log-likelihood on a validation dataset after each training iteration using the $\widehat{\mathcal{J}_{OA}}$ estimator, (4.6). For models with several hidden layers, each hidden layer was pretrained using the same hyperparameter values but only for 20 iterations, see recursive procedure in Algorithm 1.

4.5.1 Binary datasets

We start by measuring the statistical performance of a NADE trained using our order-agnostic procedure on eight binary UCI datasets (Bache and Lichman, 2013).

Experimental configuration details follow. We fixed the number of units per hidden layer to 500, following Larochelle and Murray (2011). We used minibatches of size 100. Training was run for 100 iterations, each consisting of 1000 weight updates. The initial learning rate was cross-validated for each of the datasets among values $\{0.016, 0.004, 0.001, 0.00025, 0.0000675\}$.

Algorithm 1 Pretraining of a NADE with n hidden layers on dataset X.

procedure PRETRAIN(n, X)
  if $n = 1$ **then** ▷ Recursion base case
    $nade \leftarrow$ GET-ONE-HIDDEN-LAYER-NADE-RANDOM-PARAMETERS()
    $nade \leftarrow$ TRAIN($nade, X$)
    return $nade$
  else
    $nade \leftarrow$ PRETRAIN($n-1$) ▷ Recursive call
    $nade \leftarrow$ REMOVE-OUTPUT-LAYER($nade$)
    $nade \leftarrow$ ADD-A-NEW-HIDDEN-LAYER($nade$)
    $nade \leftarrow$ ADD-A-NEW-OUTPUT-LAYER($nade$)
    $nade \leftarrow$ TRAIN($nade, X$)
    return $nade$
  end if
end procedure

Results are shown on Table 4.1 on page 75. We compare our method to mixtures of multivariate Bernoullis with their number of components cross-validated among {32, 64, 128, 256, 512, 1024}, tractable RBMs of 23 hidden units, fully visible sigmoidal Bayes networks (FVSBN), and NADEs trained using a fixed ordering of the variables. All baseline results are taken from Larochelle and Murray (2011) and details can be found there. NADEs trained in an order-agnostic manner obtain performances close to those of NADEs trained on a fixed ordering. The use of several hidden layers offers no advantage on these datasets. However, ensembles of NADEs obtain higher test log-likelihoods on all datasets.

We also present results on binarized-MNIST (Larochelle and Murray, 2011), a binary dataset of 28 by 28 pixel images of handwritten digits. Unlike classification, density estimation on this dataset remains a challenging task.

Experimental configuration details follow. Training was run for 200 iterations each consisting of 1000 parameter updates, using minibatches of size 1000. The initial learning rate was set to 0.001 and chosen manually by optimizing the validation-set log-likelihood on preliminary runs.

Results for MNIST are shown in Table 4.2 on page 79. We compare our method with mixtures of multivariate Bernoulli distributions with 10 and 500 components, fixed-ordering NADEs, RBMs (500 hidden units), and two-hidden-layer DBNs

(500 and 2000 hidden units on each layer) whose performance was estimated by Salakhutdinov and Murray (2008); Murray and Salakhutdinov (2009). In order to provide a more direct comparison to our results, we also report the performance of NADEs trained using a fixed ordering of the variables, minibatch stochastic gradient descent and sigmoid or rectified linear units. We found the type of hidden-unit did not affect statistical performance, while our minibatch SGD implementation seems to obtain slightly higher log-likelihoods than previously reported.

One and two hidden-layer NADEs trained by minimizing $\mathcal{J}_{OA}$ obtain marginally lower (worse) test-likelihoods than a NADE trained for a fixed ordering of the inputs, but still perform much better than mixtures of multivariate Bernoullis and very close to the estimated performance of RBMs. Although, lower than the estimated performance of DBNs. NADEs with more than two hidden layers are not beneficial on this dataset.

We might be tempted to conclude that the DBN's inductive bias is more useful than NADE's in modelling binary images of digits. However, it has been recently found (Murray, 2015) that the RBM and DBN used in our comparisons were trained using a slightly different dataset. Our NADEs where trained on the standard binarized-MNIST dataset (Larochelle and Murray, 2011) where the binary value of each pixel was sampled once from the original grey-scale MNIST dataset. In contrast, the pixel values were sampled for each gradient calculation when training the RBM and DBN. This difference results in an augmented dataset that has been reported to grant an advantage close to 2 nats for other models (Burda et al., 2015).

Ensembles of NADEs obtained by using NADEs with different variable orderings but trained simultaneously with our order-agnostic procedure obtain better statistical performance than NADEs trained using a fixed ordering. These EoNADEs can also surpass the estimated performance of RBMs with the same number of hidden units, and even approach the estimated performance of a (larger) 2-hidden-layer deep belief network. A more detailed account of the statistical performance of EoNADEs can be seen in Figure 4.3 on page 80. We also report the performance on NADE trained by minimizing $\mathcal{J}_{OA}$ but without input masks. Input masks are necessary for obtaining competitive results.

Samples from a 2 hidden layer (500 hidden units per layer) NADE trained using the order-agnostic method are shown in Figure 4.4 on page 81. Most of

Table 4.2: Average test-set log-likelihood per datapoint of different models on 28×28 binarized images of digits taken from MNIST.

Model	Test LogL
MoBernoullis K=10	−168.95
MoBernoullis K=500	−137.64
RBM (500 h, 25 CD steps) approx.	−86.34
DBN 2hl approx.	**−84.55**
NADE 1hl (fixed order)	−88.86
NADE 1hl (fixed order, ReLU, minibatch)	−88.33
NADE 1hl (fixed order, sigm, minibatch)	−88.35
NADE 1hl (no input masks)	−99.37
NADE 2hl (no input masks)	−95.33
NADE 1hl	−92.17
NADE 2hl	−89.17
NADE 3hl	−89.38
NADE 4hl	−89.60
EoNADE 1hl (2 orderings)	−90.69
EoNADE 1hl (128 orderings)	−87.71
EoNADE 2hl (2 orderings)	−87.96
EoNADE 2hl (128 orderings)	−85.10

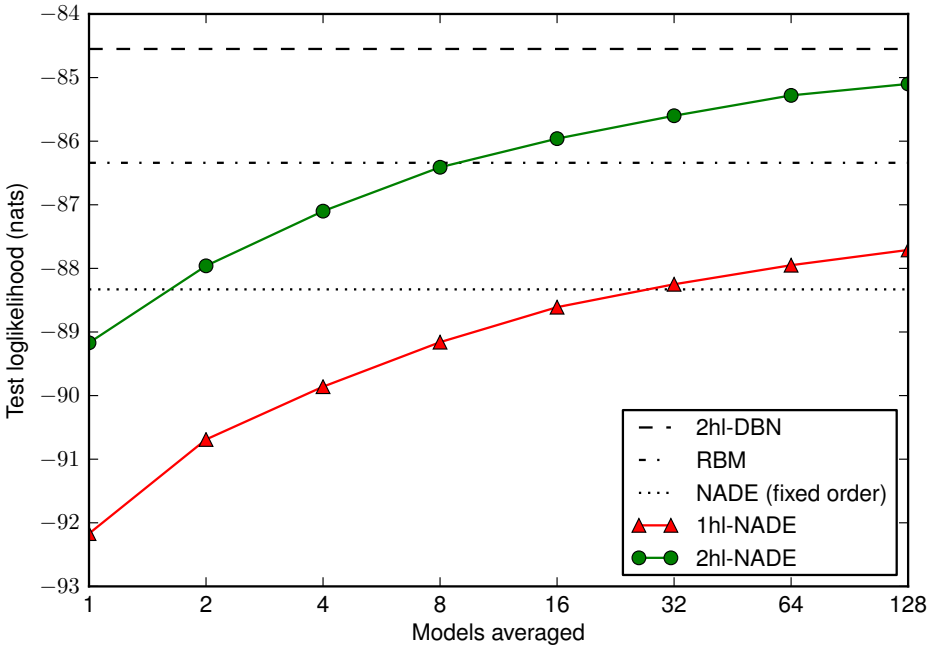

Figure 4.3: Test-set average log-likelihood per datapoint for RNADEs trained with our new procedure on binarized images of digits.

the samples can be identified as digits. In Figure 4.5 on page 82 we show 10 local modes obtained by coordinate ascent on the log-probability assigned by the model (see caption), and initialised from samples. The modes have much smoother edges than the samples and even smoother edges than the test-set datapoints in Figure 4.4.

Figure 4.7 shows some receptive fields from the model's first hidden layer (i.e. columns of $\boldsymbol{W}$). Most of the receptive fields resemble pen strokes. We also show their associated receptive fields on the input masks (i.e. columns of $\boldsymbol{U}$). These can be thought of as biases that activate or deactivate a hidden unit. Most of them will activate the unit when the input mask contains a region of unknown values (zeros in the input mask) flanked by a region of known values (ones in the input mask).

Having at our disposal a NADE for each possible ordering of the inputs makes it easy to perform any inference task. In Figure 4.6 we show examples of marginalization and imputation tasks. Arbitrarily chosen regions (10 by 10 pixels) of digits in the MNIST test-set are to be marginalized or sampled from. An RBM or a DBN would require an exponential number of operations to calculate either the marginal density or the density of the complete images. A NADE

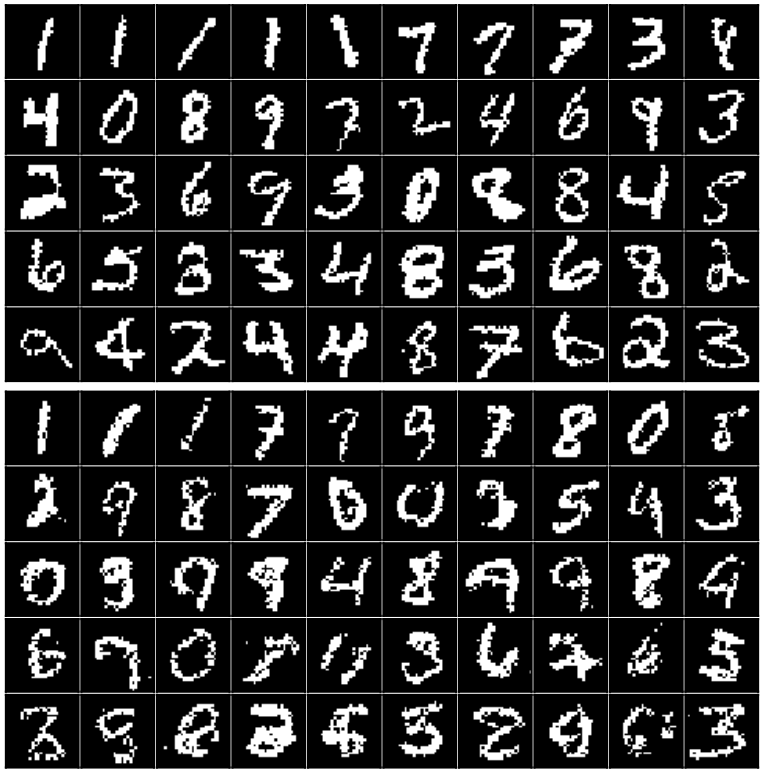

Figure 4.4: **Top:** 50 examples from binarized-MNIST ordered by decreasing likelihood under a 2-hidden-layer NADE. **Bottom:** 50 samples from a 2-hidden-layer NADE, also ordered by decreasing likelihood under the model.

trained on a fixed ordering of the variables would be able to easily calculate the densities of the complete images, but would require approximate inference to calculate the marginal densities. Both an RBM and a fixed-order NADE require MCMC methods in order to sample the hollowed regions. However, with our order-agnostic training procedure we can easily calculate the marginal densities and sample the hollowed regions in linear time just by constructing a NADE with a convenient ordering of the pixels.

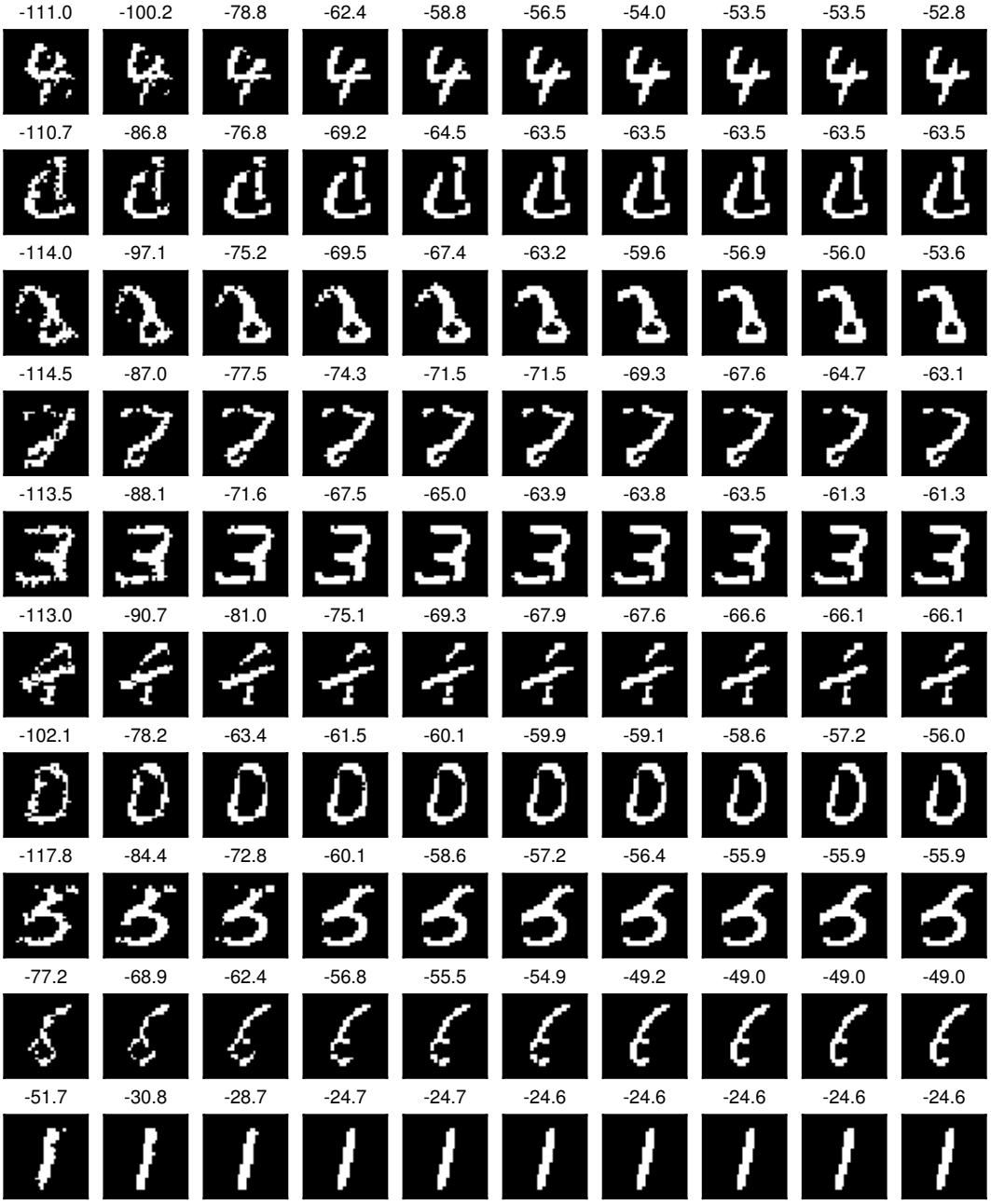

Figure 4.5: Denoising of 10 samples obtained from a 2-hidden-layer NADE by coordinate ascent on their probability mass. Pixels are taken at random and assigned the value that makes the probability of the sample higher under a fixed ordering of the dimensions. 784 pixels have been visited between each datapoint shown (left-to-right). The log-probability-mass of each datapoint is shown on top.

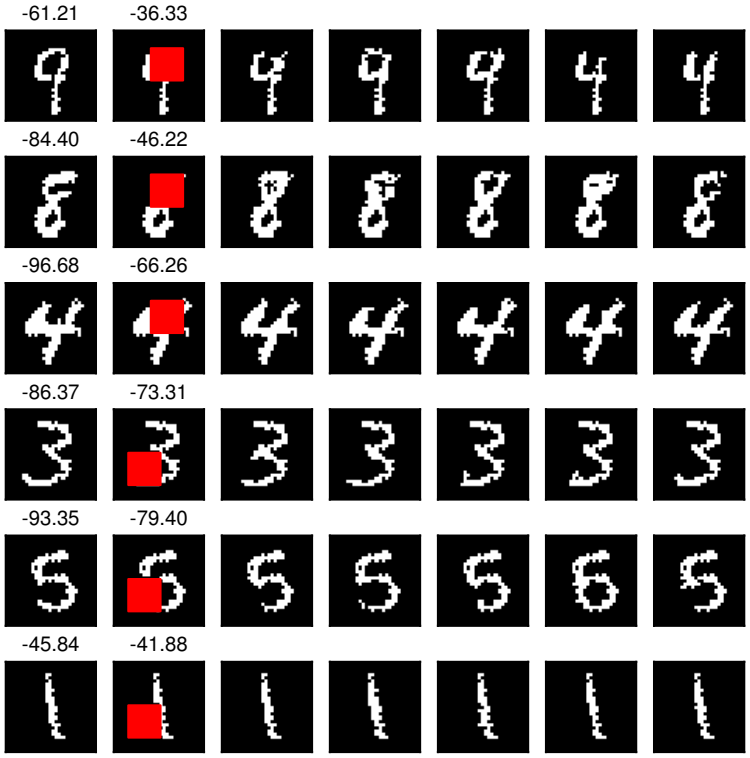

Figure 4.6: Example of marginalization and sampling. First column shows five examples from the test set of the MNIST dataset. The second column shows the density of these examples when a random 10 by 10 pixel region is marginalized. The right-most five columns show samples for the hollowed region. Both tasks can be done easily with a NADE where the pixels to marginalize are at the end of the ordering.

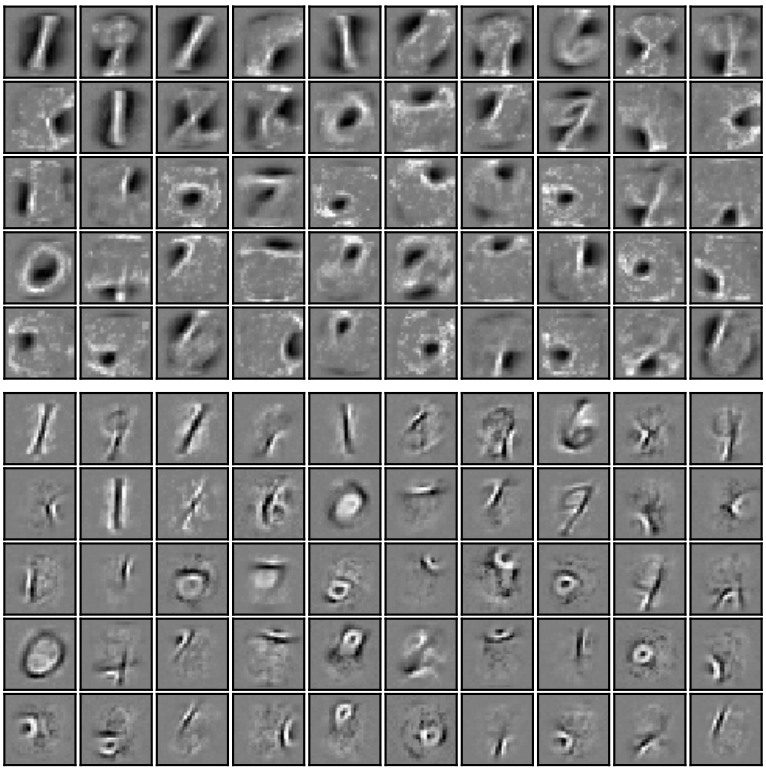

Figure 4.7: **Top:**50 receptive fields (columns of $\boldsymbol{W}$) with the biggest L2 norm. **Bottom:** Associated receptive fields to the input masks (columns of $\boldsymbol{U}$).

4.5.2 Real-valued datasets

We also compared the performance of RNADEs trained with our order-agnostic procedure to RNADEs trained for a fixed ordering. We start by comparing the performance on three low-dimensional UCI datasets (Bache and Lichman, 2013) of heterogeneous data, namely: *red wine*, *white wine* and *parkinsons*. We report the test-log-likelihood on 10 folds of the dataset, each with 90% of the data used for training and 10% for testing. All experiments use normalized data. Each dimension is normalized separately by subtracting its training-set average and dividing by its standard deviation.

Experimental details follow. Learning rate and weight decay rates were chosen by per-fold cross-validation; using grid search. One ninth of the training set examples were used for validation purposes. Once the hyperparameter values had been chosen, a final experiment was run using all the training data. In order to prevent overfitting, training was stopped when observing a training likelihood higher than the one obtained at the optimal stopping point in the corresponding validation run. All RNADEs trained had a mixture of 20 Gaussian components for output, and were trained by stochastic gradient descent on $\mathcal{J}_{OA}$. We fixed the number of hidden units to 50, as in Section 3.4.1. The learning rate was chosen among $\{0.02, 0.005, 0.002, 0.0005\}$ and the weight decay rate among $\{0.02, 0.002, 0\}$.

The results are shown in Table 4.3. RNADEs trained using our procedure obtain results close to those of RNADEs trained for a fixed ordering on the *red wine* and *white wine* datasets. On the Parkinsons dataset, RNADEs trained for a fixed ordering perform better. Ensembles of RNADEs obtained better statistical performance on the three datasets.

We also measured the performance of our new training procedure on 8 by 8 patches of natural images in the BSDS300 dataset. We compare the performance of RNADEs with different number of hidden layers trained with our procedure against a one-hidden layer RNADE trained for a fixed ordering (Section 3.4), and with mixtures of Gaussians (Zoran and Weiss, 2012).

We adopted the setup described in the previous chapter. The average intensity of each patch was subtracted from each pixel's value. After this, all datapoints lay on a 63-dimensional subspace, for this reason only 63 pixels were modelled, discarding the bottom-right pixel.

Experimental details follow. The dataset's 200 training image set was parti-

Table 4.3: Average test log-likelihood for different models on three real-valued UCI datasets. Baselines are taken from the previous chapter.

Model	Red wine	White wine	Parkinsons
Gaussian	−13.18	−13.20	−10.85
MFA	−10.19	−10.73	−1.99
RNADE (fixed)	**−9.36**	**−10.23**	**−0.90**
RNADE 1hl	−9.49	−10.35	−2.67
RNADE 2hl	−9.63	**−10.23**	−2.19
RNADE 3hl	−9.54	**−10.21**	−2.13
EoRNADE 1hl 2 ord.	−9.07	−10.03	−1.97
EoRNADE 2hl 2 ord.	−9.13	−9.84	−1.42
EoRNADE 3hl 2 ord.	−8.93	−9.79	−1.39
EoRNADE 1hl 16 ord.	−8.95	−9.94	−1.73
EoRNADE 2hl 16 ord.	−8.98	−9.69	−1.16
EoRNADE 3hl 16 ord.	**−8.76**	**−9.67**	−1.13

tioned into a training set and a validation set of 180 and 20 images respectively. Hyperparameters were chosen by preliminary manual search on the model likelihood for the validation dataset. We used a mixture of 10 Gaussian components for the output distribution of each pixel. All hidden layers were fixed to a size of 1000 units. The minibatch size was set to 1000. Training was run for 2000 iterations, each consisting of 1000 weight updates. The initial learning rate was set to 0.001. Pretraining of hidden layers was done for 50 iterations.

The results are shown in Table 4.4. RNADEs with less than 3 hidden layers trained using our order-agnostic procedure obtained lower statistical performance than a fixed-ordering NADE and a mixture of Gaussians. However RNADEs with 5 or more hidden layers are able to beat both baselines and obtain what are, to the extent of our knowledge, the best results ever reported on this task. Ensembles of RNADEs also show an improvement in statistical performance compared to the use of single RNADEs.

No signs of overfitting were observed. Even when using 6 hidden layers, the cost on the validation dataset never started increasing steadily during training. Therefore it may be possible to obtain even better results using more hidden layers or more hidden units per layer. Samples from the 6 hidden layers NADE trained in an order-agnostic manner are shown in Figure 4.8.

4.6 Discussion

In this chapter we have introduced a new training procedure that simultaneously fits a NADE for each possible ordering of the dimensions. In addition, this new training procedure is able to train deep versions of NADE with a linear increase in computation, and construct ensembles of NADEs on the fly without incurring any extra training computational cost.

NADEs trained with our procedure outperform mixture models in all datasets we have investigated. However, for most datasets several hidden layers are required to surpass or equal the performance of NADEs trained for a fixed ordering of the variables. Nonetheless, our method allows fast and exact marginalization and sampling, unlike the rest of the methods compared.

Models trained using our order-agnostic procedure obtained what are, to the best of our knowledge, the best statistical performances ever reported on the *BSDS300* 8×8-image-patches datasets. The use of ensembles of NADEs, which

Table 4.4: Average test-set log-likelihood for several models trained on 8 by 8 pixel patches of natural images taken from the BSDS300 dataset. Note that because these are log probability densities they are positive, higher is better.

Model	Test LogL
MoG $K=200$ (Zoran and Weiss, 2012)	152.8
RNADE 1hl (fixed order)	153.7
RNADE 1hl	143.2
RNADE 2hl	149.2
RNADE 3hl	152.0
RNADE 4hl	153.6
RNADE 5hl	154.7
RNADE 6hl	**155.2**
EoRNADE 6hl 2 ord.	156.0
EoRNADE 6hl 32 ord.	**157.0**

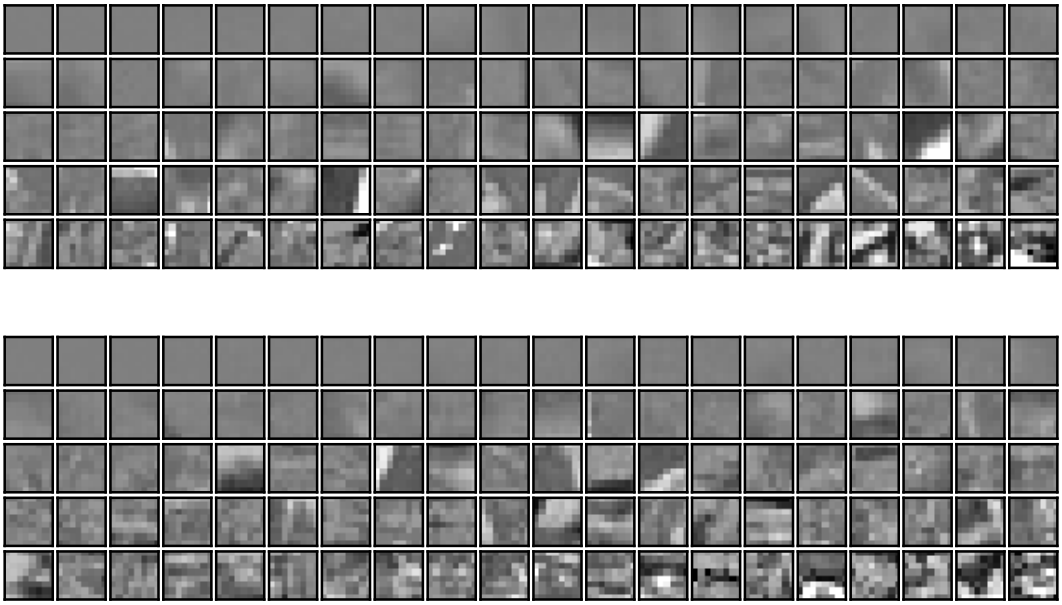

Figure 4.8: **Top:** 100 examples of 8×8 patches in the BSDS300 test set. **Bottom:** 100 samples from a 6-hidden-layer RNADE-MoG trained using our order agnostic procedure. Both sets of datapoint have been ordered by decreasing likelihood under the RNADE.

we can obtain at no extra training cost and have a mild effect on test-time cost, improved statistical performance on most datasets analyzed.

The addition of masks to indicate which of the inputs are present proved necessary in order to obtain competitive statistical performances. An explanation for their advantage is that using masks is equivalent to marginalizing the missing dimensions when each of the units in the first hidden layer is considered as a Bayes' classifier. The derivations are shown in Appendix C on page 119. The Bayes classifier resulting from two Bernoullli distributions, or two Gaussians with the same covariance matrix, is equivalent to a sigmoid-linear discriminant (MacKay, 2003). Neural networks usually assume that all input features are present at all times, and therefore the parameterization in terms of the discriminant function is preferred as it has a smaller number of parameters and, unlike the Bayes classifier parameterization, it does not suffer from aliasing (for a single unit). However, in cases like the order-agnostic NADE, it may be advantageous to maintain the full Bayes' classifier parameterization so that marginalization of missing inputs can be performed. This idea might also be worth exploring in other models, like denoising autoencoders with "drop noise", or dropouts regularization.

It is perhaps unsurprising that ensembles obtained higher statistical performance than single NADEs, given that we chose the ordering of the single NADEs at random. Due to the concavity of the log function, Jensen's inequality guarantees that $\mathop{\mathbb{E}}_{o} \log p(\boldsymbol{x} \,|\, o) < \log \mathop{\mathbb{E}}_{o} p(\boldsymbol{x} \,|\, o)$, which does not preclude a single ordering from being better than an ensemble. In all our experiments using ensembles, we chose the orderings of variables for each of the components at random, but more elaborate techniques could be used (Smyth and Wolpert, 1999).

Chapter 5

RNADE for speech synthesis

This chapter is an extended version of the article "Modelling acoustic feature dependencies with artificial neural networks: Trajectory-RNADE" (Uria et al., 2015) published in ICASSP 2015.

Given a transcription, sampling from a good model of acoustic feature trajectories should result in plausible realizations of an utterance. However, samples from current probabilistic speech synthesis systems result in low quality synthetic speech. Henter et al. have demonstrated the need to capture the dependencies between acoustic features conditioned on the phonetic labels in order to obtain high quality synthetic speech. These dependencies are often ignored in neural network based acoustic models (Section 5.2). We tackle this deficiency by introducing a probabilistic neural network model of acoustic trajectories, trajectory RNADE, able to capture these dependencies (Sections 5.3, 5.4, 5.5). Trajectory-RNADE produces higher quality mean and sampled trajectories than those generated by a mixture density network (Section 5.6).

5.1 Introduction

We define *text to speech synthesis* (TTS), or simply *speech synthesis*, as the task of generating an acoustic waveform intelligible by a human listener given a text transcription.

Speech synthesis technology has high commercial value. It is regularly used by automated phone systems, navigation systems and virtual digital assistants among others.

Good synthetic speech must be intelligible, but other attributes like natural-

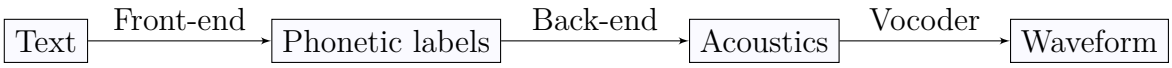

Figure 5.1: Speech generation diagram.

ness, speaker identifiability, and the ability to convey the emotional state of the purported speaker are also important.

A schematic diagram showing the functioning of a typical TTS system can be seen in Figure 5.1. Generation of synthetic speech occurs in three stages. In the first stage, text is input to the system and analyzed by the *front-end* which generates phonetic labels for each frame of speech (typically separated by 5 milliseconds). The number of phonetic labels in a modern system can run in the hundreds and includes information about the identity of the phone, stress level, tone, grammatical category, distance to the end of word, and others. In a second stage, the *back-end* generates acoustic features that correspond to the phonetic-labels output by the front-end. In a third stage the acoustic features are transformed into a waveform using the appropriate vocoder. All three stages are active fields of research. In this chapter we try to improve the back-end stage by utilizing a conditional version of RNADE to model the distribution of the acoustic features given the phonetic labels.

Speech synthesis systems can be classified in two types depending on whether they use a unit-selection or a probabilistic back-end. Unit-selection systems (Hunt and Black, 1996) store a database of natural speech and generate synthetic speech by pasting together segments of natural speech whose phonetic labels are most similar to those of the text to be synthesized. In contrast, probabilistic back-ends use a probabilistic model of the acoustics conditioned on the phonetic labels. The most common kind of probabilistic model is a diagonal-covariance Gaussian whose mean is calculated using a decision tree (Zen et al., 2009).

Speech generated by modern TTS systems of either kind is highly intelligible (Black and Tokuda, 2005). Probabilistic back-ends tend to be more amenable to emotion and speaker adaptation, can rely on smaller training datasets, and have a smaller memory footprint in exchange for more computation. However, unit-selection systems with big unit reservoirs (tens of hours) still result in higher quality synthetic speech (Zen et al., 2009).

In this chapter we aim to improve the quality of synthetic speech generated

by probabilistic back-ends by using a conditional version of RNADE instead of mixture density networks (Zen and Senior, 2014) or decision-tree-tied Gaussian models (Zen et al., 2009).

5.1.1 Dependencies among acoustic features

Given a text transcription, samples from a good conditional probabilistic model of the acoustics should result in plausible speech acoustic realizations. However, samples from current probabilistic models sound noisy and unnatural (Shannon et al., 2011). For this reason, it is common practice to output the mean acoustic trajectory when synthesising speech. However, mean acoustic trajectories sound muffled due to their unusually high smoothness. To reduce this over-smoothing, postfiltering and generation techniques that take into account the variance of the acoustic trajectory are usually applied to modify the acoustics generated (Tomoki and Tokuda, 2007).

Two incorrect conditional independence assumptions contribute to the unnaturalness of samples from traditional acoustic models: conditional independence across time; and conditional independence across acoustic features.

Conditional independence across time is usually dealt with by augmenting the acoustic features with dynamical features, i.e. finite differences in time (Tokuda et al., 2000). Using these dynamical features, it is possible to construct a joint probabilistic model of the trajectory across time (Zen et al., 2004, 2007b). We will review the trajectory-HMM in Section 5.4.

In contrast, conditional independence across acoustic features is often taken for granted in speech synthesis systems (Zen et al., 2013; Zen and Senior, 2014; Qian et al., 2014). However, experiments carried out by Henter et al. (2014) indicate that ignoring the dependency between acoustic features conditioned on a transcription, results in lower perceived naturalness. In their experiments, Henter *et al.* recorded several instances of the same utterance produced by a single speaker. Using this database, they produced resampled instances by using trajectories from different recordings for each of the acoustic features. These resampled instances would have, on average, the same probability density under the data generating distribution if the conditional independence assumption was valid. The resampled instances were compared to the original recordings by a

panel of non-expert listeners using a sensitive MUSHRA[1] (ITU, 2003) assessment methodology. The results pointed to a significant decrease of perceived quality in the resampled recordings.

In traditional decision-tree-tied Gaussian models, independence across features can be relaxed by the use of full (Shannon et al., 2011) or semi-tied covariance matrices (Gales, 1999). Here, we attempt to tackle these conditional dependencies across acoustic features in systems based on artificial neural networks, where it has commonly been ignored.

5.2 Artificial neural networks for acoustic modelling

Artificial neural networks for regression can be interpreted as conditional probability models. The output of a neural network trained to minimise the mean square error criterion, can be interpreted as the mean of a fixed variance Gaussian distribution trained to maximize the conditional probability of the output given the input (Bridle, 1990).

Mixture density networks (MDNs) (Bishop, 1994, 1995) provide an explicit and more powerful model of conditional probability distributions. An MDN uses a neural network to output the parameters of a fixed family of distributions (for example means, variances, and component weights for a mixture of Gaussians) given some inputs. MDNs have been used for speech synthesis (Zen and Senior, 2014), modelling the conditional probability of acoustic features conditioned on phonetic labels.

Usually MDNs output the parameters of a one-dimensional mixture of Gaussians for each dimension, in which case, the model assumes conditional independence across dimensions given the input, see Figure 5.2.a. An MDN could be designed to output a full covariance matrix, for example by outputting the parameters of its Cholesky decomposition (Williams, 1996), but that approach will not scale past a few dimensions, given that the number of outputs would grow quadratically with the dimensionality of the acoustic features.

[1] MUltiple Stimuli with Hidden Reference and Anchor

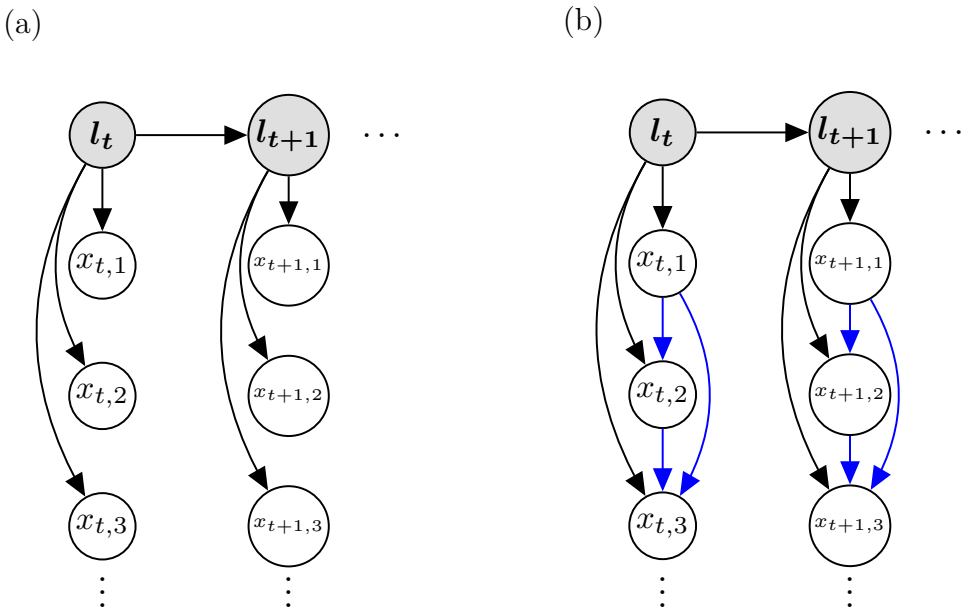

Figure 5.2: (a) Hidden Markov model with observations (acoustic features) conditionally independent of each other given the hidden state (phonetic labels). This conditional independence assumption is present in mixture density networks and diagonal-covariance Gaussian models. (b) Hidden Markov model with no conditional independence assumption between dimensions in the same time frame. Conditional RNADE introduces the arrows shown in blue with respect to a mixture density network. This graphical model can also be implemented by a hierarchical Markov decision tree (Jordan et al., 1997).

5.3 Conditional RNADE

To capture the dependencies among acoustic features, we propose the use of RNADE to capture the joint conditional distribution of acoustic features, $\boldsymbol{x}_t$, conditioned on phonetic labels, $\boldsymbol{l}_t$. Designing a conditional version of RNADE (or binary NADE) can be done by adding another set of inputs that are always present in the calculation of the one-dimensional conditionals.

As we have seen in Chapter 4, deep RNADEs can be efficiently trained, but sampling has $O(DH^2)$ computational complexity, which makes them too slow for speech synthesis, where computational performance is crucial. Therefore we will limit our investigations to RNADEs with one hidden layer and fixed ordering, i.e. the class of models introduced in Chapter 3. The practical limitation to one hidden layer only affects the autoregressive part of RNADE. It is possible to use several hidden layers to compute useful predictive features from the phonetic labels, as shown in Figure 5.3.

In order to also capture dependencies across time, we combine the trajectory-HMM formulation with a conditional-RNADE, a model we will call trajectory-RNADE. In the next section we review the trajectory-HMM, after which we provide a more detailed description of trajectory-RNADE.

5.4 Trajectory hidden Markov models

Trajectory hidden Markov models (Zen et al., 2004) relax the conditional independence assumption between observations at different times given the latent variable in HMMs. To do so, trajectory-HMMs model not only the value of the observations, given the latent variables, but also the first and second derivatives of the observations. In practice, given that the observations are measured at discrete times, finite differences between the observation and its preceding and following values are used. These finite differences (or deltas in speech-synthesis parlance) are commonly calculated using equations:

$$\Delta x_t = -\frac{1}{2}x_{t-1} + \frac{1}{2}x_{t+1} \tag{5.1}$$

$$\Delta^2 x_t = x_{t-1} - 2x_t + x_{t+1} \tag{5.2}$$

For simplicity we will limit the exposition to one-dimensional observations (acoustic features), the extension to multivariate observations is straightforward.

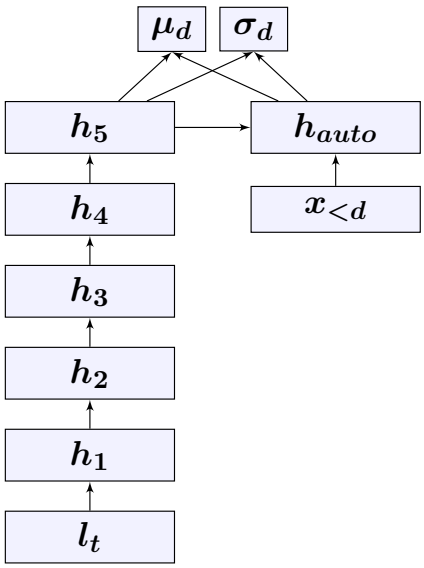

Figure 5.3: Connectivity in a conditional RNADE with one hidden autoregressive layer and several conditional feature extraction layers. The diagram only shows the outputs used when predicting the parameters of a Gaussian distribution over the d-th acoustic feature, $x_{t,d}$, conditioned on phonetic labels, $\boldsymbol{l_t}$, and other acoustic features already predicted, $\boldsymbol{x}_{<d}$. At training time the values of $\boldsymbol{x}_{<d}$ are taken from the training dataset, at test time they are samples obtained from the network.

Let us denote by $\boldsymbol{x} \equiv (x_1, x_2, \ldots, x_T)$ the vector of observations, and by

$$\boldsymbol{z} \equiv (x_1, x_2, \ldots, x_T, \Delta x_1, \Delta x_2, \ldots, \Delta x_T, \Delta^2 x_1, \Delta^2 x_2, \ldots, \Delta^2 x_T) \tag{5.3}$$

the *augmented vector* of observations and deltas, and by $\boldsymbol{l} \equiv (\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_T)$ the vector of phonetic labels for a sequence of length T. Given the vector of labels, a trajectory-HMM model assumes conditional independence between the observations across time, and between the observations and the deltas:

$$p_z(\boldsymbol{z} \,|\, \boldsymbol{l}) = \prod_{t=1}^{T} p(x_t \,|\, l_t) p(\Delta x_t \,|\, l_t) p(\Delta^2 x_t \,|\, l_t) \tag{5.4}$$

However, given (5.1) and (5.2), any *plausible* augmented vector can be obtained as a linear transformation of a vector of static observations, $\boldsymbol{z} = \boldsymbol{M}\boldsymbol{x}$, where the matrix M of size $3T \times T$ has the form:

$$\boldsymbol{M} = \begin{pmatrix} 1 & 0 & 0 & 0 & & & \\ 0 & 1 & 0 & 0 & & & \\ 0 & 0 & 1 & 0 & & & \\ & & & \ddots & & & \\ & & & & 0 & 1 & 0 \\ & & & & 0 & 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} & 0 & 0 & & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 & & & \\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & & & \\ & & & \ddots & & & \\ & & & & -\frac{1}{2} & 0 & \frac{1}{2} \\ & & & & 0 & -\frac{1}{2} & \frac{1}{2} \\ -1 & 1 & 0 & 0 & & & \\ 1 & -2 & 1 & 0 & & & \\ 0 & 1 & -2 & 1 & & & \\ & & & \ddots & & & \\ & & & & 1 & -2 & 1 \\ & & & & 0 & 1 & -1 \end{pmatrix} \tag{5.5}$$

That is, all plausible augmented observations lie on the T-dimensional subspace spanned by the columns of this matrix. If we ignore all implausible augmented trajectories outside of this subspace, the distribution over observations, $\boldsymbol{x}$, will follow a renormalized distribution:

$$p_x(\boldsymbol{x}) = \frac{1}{K} p_z(\boldsymbol{M}\boldsymbol{x}) \qquad \text{where} \qquad K = \int p_z(\boldsymbol{M}\boldsymbol{x}) d^T\boldsymbol{x} \tag{5.6}$$

When the distribution over augmented vectors is modelled by a Gaussian, $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu_z}, \boldsymbol{\Sigma_z})$, it is straightforward to calculate the renormalized probability over trajectories of observations. The distribution over the subspace of $\boldsymbol{z}$ that satisfies constraints (5.1) and (5.2) is also a Gaussian, whose distribution can be found completing the square:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu_x}, \boldsymbol{\Sigma_x}) \tag{5.7}$$

$$\boldsymbol{\Sigma_x} = \left(\boldsymbol{M}^\top {\boldsymbol{\Sigma_z}}^{-1} \boldsymbol{M}\right)^{-1} \tag{5.8}$$

$$\boldsymbol{\mu_x} = \boldsymbol{\Sigma_x} \boldsymbol{M}^\top {\boldsymbol{\Sigma_z}}^{-1} \boldsymbol{\mu_z} \tag{5.9}$$

Even if $\boldsymbol{\Sigma}_{\boldsymbol{z}}$ is diagonal, as we assumed in (5.4), the resulting $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ will not, in general, be diagonal, and will, in effect, capture the dependency of the observations across time. Under this model, $\boldsymbol{\mu}_{\boldsymbol{x}}$ is the most probable trajectory, also called the MLPG-trajectory (maximum likelihood parameter generation) by speech synthesis practitioners.

It is possible to approximate the MLPG calculations for a mixture of Gaussians distributions over each frame of augmented observations (Zen et al., 2004). However, these methods only guarantee convergence to a local maximum. Sampling with MoG marginals over augmented vectors would also require approximate inference techniques.

5.5 Trajectory-HMMs with RNADE observations

In order to capture dependencies across acoustic features in the same time frame and dependencies across time, we introduce the trajectory-RNADE hidden Markov model. We define a trajectory-RNADE as a conditional-RNADE that models the acoustic features of a single time frame autoregressively, doing it in groups of three dimensions at a time (observation, delta and double delta).

In order to sample an utterance, the model outputs the distribution over the observation and deltas of a single acoustic feature for each time frame conditioned on their corresponding phonetic labels. Then a trajectory for that acoustic feature is sampled (or the most probable trajectory is chosen) using the trajectory-HMM formulation shown in the previous section. When the trajectory of an acoustic feature has been decided, it can be used as input to the autoregressive networks that predict the trajectory of subsequent features. See Algorithm 2.

In order to be able to use the trajectory-HMM formulation, we used a single univariate Gaussian as the conditional distribution of each acoustic feature (or delta) in all our experiments.

In a trajectory RNADE the trajectory of each acoustic feature over the whole utterance is predicted before the trajectories of the following acoustic features are predicted.

As it is common, during training we model the static and delta features as unconstrained variables, even though the trajectory can only lie on a subspace of the augmented space. This inconsistency between the training criterion and the actual generation procedure causes an underestimation of the variability of the

Algorithm 2 Sampling from a trajectory-RNADE HMM conditioned on phonetic labels. The utterance has a duration of T frames and the acoustic features have dimensionality D.

procedure SAMPLE($\boldsymbol{l}_1, \ldots, \boldsymbol{l}_T$)
  x = array$[D \times T]$ ▷ acoustics
  for $d = 1 \ldots D$ **do**
    μ_z = array$[3T]$
    Σ_z = array$[3T]$ ▷ diagonal
    for $t = 1 \ldots T$ **do**
      $\mu_z[t, t+T, t+2T]$, $\Sigma_z[t, t+T, t+2T]$ = RNADE(l_t, x$[1 \ldots (d-1)$, $t]$)
    end for
    μ_x, Σ_x = DISTRIBUTION-OVER-OBSERVATIONS(μ_z, Σ_z)
      ▷ Implemented by Equations (5.8) and (5.9)
    x$[d, \cdot]$ = SAMPLE-GAUSSIAN(μ_x,Σ_x)
  end for
  return x
end procedure

acoustic trajectory (Shannon et al., 2011).

The trajectory HMM formalism corresponds to a product of Gaussians (Williams, 2005). A product of Gaussians is itself a Gaussian distribution whose precision parameter is the sum of the precisions of the n Gaussians multiplied. If we assume the Gaussians multiplied have approximately the same variance, by multiplying each of their variances by n, their product will have the same variance. This leads to a heuristic where we will multiply the variances of the static and delta features by 3. This heuristic has previously been shown to improve the likelihoods of trajectory models (Shannon et al., 2011, 2013), and our results in the next section agree with this finding. A training technique consistent with the trajectory model generation and practically applicable to neural networks has been recently developed (Xie et al., 2014) in the field of voice conversion, and could be incorporated to our experiments.

5.6 Experimental design

Our experimental goal is to compare the statistical performance of trajectory-HMMs with observations modelled by either conditional-RNADEs or by MDNs.

The systems were trained using the database recorded to build the TTS entry by Cooke et al. (2013). It consists of 2 hours of data of a British male voice. The data used here was sampled at 48kHz. We split the dataset into a training subset of 2602 utterances, 98 validation utterances (used for early stopping and selecting hyperparameters during training) and a test set of 99 utterances. We used STRAIGHT vocoding (Kawahara et al., 1999) to extract 60 mel-cepstral features, 25 band aperiodicity features, f_0 (linearly interpolated in unvoiced regions) and a voiced/unvoiced feature for a total of 87 dimensions. In order to train both the RNADE and the MDN we used forced-alignments obtained with a standard HMM-GMM system (Zen et al., 2007a).

The input phonetic labels were the same for both the MDN and RNADE. We included the substate alignment from the HMM-GMM system and its relative position within the substate (Qian et al., 2014).

The acoustic data was rescaled to the range 0.01–0.99, logit-transformed, augmented with deltas and double deltas, and standardized to mean zero and standard deviation one. Modelling the logit-transformed version of the data guarantees that samples will remain in an acceptable range even when using Gaussian models, which have infinite support.

The MDN used in our experiments had five hidden layers, each had 600 rectified linear units (Nair and Hinton, 2010). We used a Gaussian output per dimension (mean and standard deviation) as we found no increase in statistical performance (as measured by log-likelihood of the model for a validation dataset) when using a mixture of Gaussians (MoG). Previous work has reported a benefit from mixture models (Zen and Senior, 2014), although those models were fitted with ten times more data than here.

The trajectory RNADE in our experiments also had five conditional hidden layers, and one autoregressive hidden layer. Each hidden layer had 600 rectified linear units (see Figure 5.3). We used a Gaussian output per dimension, again finding no increase in log-likelihood for a validation dataset when using a MoG. In the RNADE model we chose the following order for the acoustic features: voiced/unvoiced, f_0, mel-cepstral features (0 to 59 in that order), band

aperiodicities (1 to 25 in that order).

Both models were trained using AdaDec (Senior et al., 2013) for 1000 epochs of 1000 updates each, minibatches of size 100 were used. The learning rate was initialized to 3×10^{-4} and decreased by 3×10^{-7} after each epoch.

5.7 Experimental results

We are mainly interested in measuring the quality of the two systems as generative models of speech acoustics conditioned on phonetic labels. Samples and mean trajectories from the two models are available online[2]. To measure the statistical performance of the models we report their test-set log-likelihood (average log-density of a frame in the held out test set). Results are shown in Table 5.1. The first row shows the log-likelihood of each model considering the static and delta features as unconstrained dimensions, this is our training criterion. Note that the first row is not comparable to the rest, as it considers models over the delta-augmented acoustic (261 dimensions) space, while the following rows consider only the acoustic trajectories (87 dimensions). The second row shows the likelihoods of the models by ignoring the delta-features, i.e. each frame is considered independent of the rest conditioned on the labels. The third and fourth rows show the likelihood of the models using the trajectory HMM formalism to calculate the density of the trajectories.

To compensate for the underestimation in variance caused by a training criterion inconsistent with the trajectory model (the delta-features are predicted independently of the static features, even if they can be deterministically calculated given all static values), we also calculated the densities under a trajectory model where the variances of static and delta features were multiplied by 3 (we also tried multiplying the variances by other factors but they resulted in lower test log-likelihoods). As commented in Section 5.5, this heuristic increase in variance improves the likelihoods and the global variance of samples.

Trajectory-NADE achieved higher likelihoods under all criteria. This leads us to conclude that it is a better joint model of acoustic feature dependencies conditioned on phonetic labels than an MDN.

Due to the sequential nature of RNADE we can calculate which acoustic features are being predicted more accurately than by an MDN. In Figure 5.4 we

[2] http://www.benignouria.com/permalink/rnade_synthesis

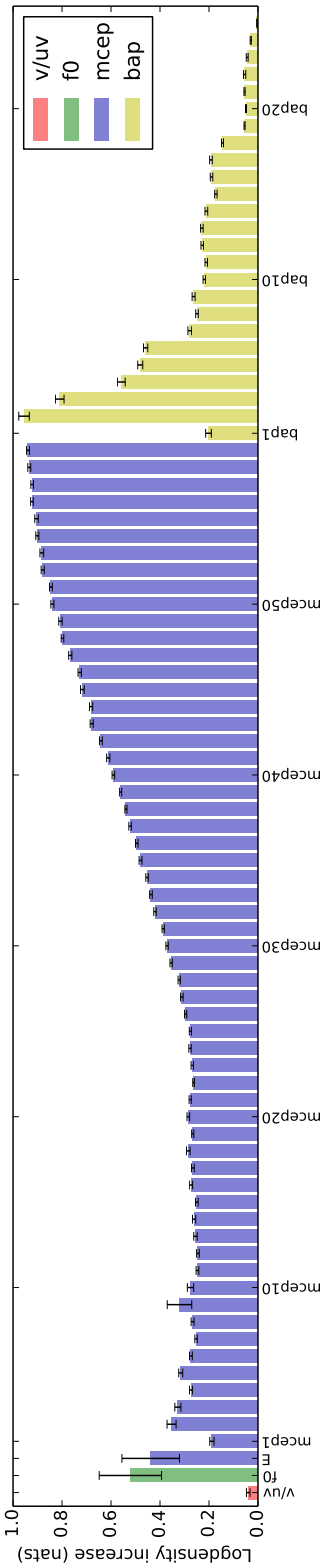

Figure 5.4: Increase in average log-density per acoustic feature obtained by a trajectory RNADE with respect to a trajectory MDN. Both models had the variances of the static and delta features multiplied by 3 before calculating the distribution over trajectories. Dimensions are shown in the order they are predicted by the RNADE model.

Criterion	MDN	Traj-RNADE
Avg. log-density x, Δx, $\Delta\Delta x$	0.87	102.60
Avg. log-density x	-32.45	7.18
Avg. log-density x trajectory model	-18.16	23.04
Avg. log-density x trajectory model ($3\times$ variance)	23.12	**59.27**

Table 5.1: Average log-density per frame (in nats) on a held-out test set of 96 utterances, greater numbers are indicative of better models.

investigate the source of the increase in likelihood. The y-axis shows the average increase in likelihood obtained by modelling the trajectory of a feature using a trajectory RNADE instead of a trajectory MDN (both having the variances of statics and deltas multiplied by 3, bottom row in Table 5.1). The figure shows the features in the order they are predicted by the RNADE model. We expect no increase in prediction accuracy of the voiced/unvoiced feature, as the RNADE uses uses the same predictor variables as the MDN in this case (any difference is caused by the stochastic training procedure). Regarding f_0, we observe an increase in accuracy of about half a nat, which shows that knowing whether a frame is voiced or not is helpful in predicting f_0 (the f_0 values were linearly interpolated in unvoiced regions). A similar increase in log-density is observed in the prediction of the frame energy (mcep_0). Regarding the mel-cepstral features, there is a slight increase in log-density of the 1st feature (mcep_1), followed by a higher increase in the middle cepstral features (mcep_2–mcep_{28}) and a greater and growing increase in accuracy in the higher cepstral features (mcep_{29}–mcep_{59}) that describe the fine structure of the spectrum. These features are very difficult to predict conditioned only on the phonetic labels, but seem to show a high degree of dependency with each other. Regarding band aperiodicity features, we again find a small increase in the log-density of the first feature followed by a higher increase in the following features (which are highly correlated to the first).

We are also interested in measuring the quality of the synthetic speech generated by each system. We performed three forced-preference tests comparing: (1) trajectory samples from an MDN and an RNADE, (2) trajectory means from an MDN and an RNADE (3) trajectory samples and trajectory means from an RNADE. Samples were obtained using the $3\times$ variance heuristic, and means

MDN		Traj. RNADE		n
Sample	Mean	Sample	Mean	
19.4%	-	**80.6%**	-	900
-	-	16.0%	**84.0%**	865
-	33.6%	-	**66.4%**	794

Table 5.2: Subjective evaluation results. The last column shows the total number of comparisons performed by the 29 participants for each of the tests. For statistically significant results at a 0.99 level in a two-tailed binomial test (indiference null-hypothesis) the preferred system is shown in bold font.

using the MLPG (Tokuda et al., 2000) algorithm. All tests were performed by a group of 29 native English speakers using headphones in sound-deadened booths. The participants were asked to choose the higher quality instance from pairs of utterances presented in a random order. The results can be seen in Table 5.2. The participants showed preference for means and samples generated from the RNADE instead of the MDN. They also showed preference for means instead of samples generated by RNADE.

5.8 Discussion

The use of NADE for speech synthesis was proposed before by Yin *et al.* (Yin et al., 2014). They used fixed-variance real-valued version of NADE to replace the means of an HMM-GMM speech synthesis system; training a different NADE for each state. In contrast, our approach takes full advantage of the neural network formulation of RNADE: it uses a single network that takes phonetic labels as inputs, and outputs means and standard deviations for each acoustic feature.

The trajectory HMM formulation, using deltas to model the dependencies across time, is a limiting one. Calculating the distribution over trajectories is slow. It requires a matrix inversion, making it difficult to optimize the right training criterion. It also limits the family of distributions that can be used to model each frame of the augmented-feature distribution to a Gaussian. Autoregressive HMMs are a promising research direction for speech synthesis (Shannon et al., 2013), but in preliminary experiments training fully-autoregressive RNADEs (autoregressive

both across the features in a frame and across time) got stuck in local optima that model smooth speech-like acoustic trajectories but ignore the phonetic labels.

We will also investigate the importance of the order in which the acoustic-dimensions are predicted. RNADEs with different orders may model different joint distributions, especially if any of the empirical one-dimensional conditionals is multimodal.

In conclusion, our experimental results show that trajectory RNADE is better than an MDN at modelling the joint distribution of acoustic features conditioned on phonetic labels. Furthermore, trajectory RNADE also produces higher quality synthetic speech as judged by subjective preference tests.

Chapter 6

Conclusions

In this thesis, we have presented RNADE, the first formulation of a neural autoregressive model for real-valued data. We have shown RNADE is a capable model that obtained state-of-the-art statistical performance for speech acoustics, and small natural image patches. RNADE also has attractive computational characteristics, namely, probability density estimation and sampling of complete datapoints scales with the square of the data dimensionality (assuming the number of hidden units is proportional to the dimensionality of the data). This allows for easy training by stochastic gradient descent on the negative log-likelihood for a training dataset, and makes model comparison with other tractable models possible.

We have also presented a training procedure that circumvents the main drawback of autoregressive models: the fixed ordering of the variables modelled. Using our order-agnostic training procedure, it is possible to obtain a set of parameters that can be used by a NADE (or RNADE) for any ordering of the variables. Choosing the order of the variables at test-time allows us to obtain a model that can marginalize or sample any subset of dimensions exactly and tractably.

The same training procedure makes it practical to train NADEs with several hidden layers. These *deep* models obtained even higher statistical performance than shallow NADEs or RNADEs for some natural image patches and segments of speech spectrograms.

We have also presented a method to obtain ensembles of NADEs. Our ensembles of NADEs are mixtures of NADEs that share the same parameter values, but differ in their ordering of the variables. These ensembles can be created with no extra training computation by using our order-agnostic training. The ensembles

obtained higher test-likelihoods than single NADEs in all datasets tested.

Perhaps most importantly, some of the work presented in this thesis has already proved useful to other researchers. RNADE has been used to improve the mixing rate of generalized denoising autoencoders by Ozair et al. (2013). As we briefly commented in Section 2.7, a necessary condition for the consistency of a GSN is the need for the reconstruction operator to be consistent for the true conditional reconstruction distribution. Often, this reconstruction operator must be able to capture multimodalities, and sampling new reconstructions must be fast to allow for better mixing, both traits make conditional-RNADE an ideal candidate.

Our work on order-agnostic training, and particularly the ability to train a deep NADE has been used by Raiko et al. (2014) to create NADE-k, which extends the 1-step mean field approximation of RBMs that inspired NADE to k steps. A NADE-k can be seen as a deep-NADE with tied weights, where every even layer reconstructs the input and every odd layer recomputes the hidden units. NADE-k obtains slightly better likelihoods than a regular NADE on the datasets tested by Raiko et al.. However, as with every deep NADE, this increased performance comes at the price of cubic complexity for sampling and density estimation.

Regular binary NADEs have been used as a variational distribution for the posterior of binary latent variables in Helmholtz machines in the so called reweighted wake-sleep algorithm (Bornschein and Bengio, 2014). RNADE makes an extension to real-valued latent variables straightforward.

The work detailed in this thesis can be extended in numerous ways. The choice of a parametric form for the conditionals of an RNADE can be rather arbitrary in the absence of domain knowledge. A possible extension would be the use of quantile regression (Koenker and Bassett, 1978) and monotonic interpolation to approximate a free form cumulative distribution function for each one-dimensional conditional, which would also facilitate its use for compression. Another aspect that requires further investigation is the use of fully autoregressive (across time and acoustic features) conditional RNADEs for speech synthesis. The optimization problems we found in our initial attempts may be tackled using training techniques that condition on previous samples of the model (Bengio et al., 2015).

One of the most interesting aspects of NADE is its ability to leverage neural-network techniques to obtain tractable density estimators. The field of artificial-neural-networks is blooming with new advances every year. By adapting these advances to the NADE and RNADE framework, we can obtain more power-

ful density estimators[1]. Active fields of research include: the design of new non-linearities (Nair and Hinton, 2010; Goodfellow et al., 2013), regularization techniques (Srivastava et al., 2014; Goroshin and LeCun, 2013), new optimization techniques (Martens, 2010; Schaul et al., 2013), and hyperparameter selection procedures (Snoek et al., 2012, 2015). All of these techniques became prominent after the initiation of this thesis and can conceivably be incorporated into our models, hopefully leading to even better statistical performance than the one we have shown.

In the last chapter of this thesis, we have demonstrated the applicability of the techniques presented to improve the results of a speech synthesis system, a real-world task of high commercial value. It is our hope that the work presented here will soon prove useful in many other commercial and scientific endeavours.

[1] Assuming that the inductive biases that prove useful for discrimination are also useful for unsupervised modelling.

Appendix A

Mean-field Gaussian-RBM derivations

In this appendix we derive the equations for the mean-field approximation of the autoregressive conditionals of a Gaussian-RBM. No new material is presented.

We are interested in approximating $p(x_d \,|\, \boldsymbol{x}_{<d})$. To do so, we will calculate the mean-field approximation of $p(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d})$:

$$p(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d}) = \frac{1}{Z(\boldsymbol{x}_{<d})} \exp\left\{ \sum_{j=d}^{D} \frac{(x_j - b_j)^2}{2\sigma_j^2} - \sum_{k=1}^{H} h_k c_k - \sum_{k=1}^{H} \sum_{j=1}^{D} \frac{x_j}{\sigma_j} W_{j,k} h_k \right\} \tag{A.1}$$

by a factorial $q(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d})$ from which it is trivial to marginalize $q(x_d \,|\, \boldsymbol{x}_{<d})$:

$$q(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d}) = \prod_{j=d}^{D} \mathcal{N}(x \,|\, \mu_j, \delta_j^2) \prod_{k=1}^{H} Bern(h \,|\, \tau_k) \tag{A.2}$$

$$= \prod_{j=d}^{D} \frac{1}{\sqrt{2\pi}\delta_j} \exp\left\{ -\frac{1}{2} \frac{(x_j - \mu_j)^2}{\delta_j^2} \right\} \prod_{k=1}^{H} \tau_k^{h_k} (1 - \tau_k)^{(1 - h_k)} \tag{A.3}$$

$$\tag{A.4}$$

We will minimize the $\mathrm{KL}\,(q \,\|\, p)$ with respect to the parameters of q: μ, δ, τ:

$$\begin{aligned}
\mathrm{KL}\left(q \parallel p\right) =& \left\langle -\log p(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d}) \right\rangle_{\boldsymbol{x}_{\geq d}, \boldsymbol{h} \sim q} - \left\langle -\log q(\boldsymbol{x}_{\geq d}, \boldsymbol{h} \,|\, \boldsymbol{x}_{<d}) \right\rangle_{\boldsymbol{x}_{\geq d}, \boldsymbol{h} \sim q} && \text{(A.5)} \\
=& K(\boldsymbol{x}_{<d}) + \sum_{j=d}^{D} \frac{1}{2\sigma_j^2} \left\langle x_j^2 \right\rangle_q - \sum_{j=d}^{D} \frac{b_j}{\sigma_j^2} \langle x_j \rangle_q - \sum_{k=1}^{H} c_k \langle h_k \rangle_q - \\
& - \sum_{j=1}^{d-1} \sum_{k=1}^{H} \frac{1}{\sigma_j} x_j W_{j,k} \langle h_k \rangle_q - \sum_{j=d}^{D} \sum_{k=1}^{H} \frac{1}{\sigma_j} \langle x_j \rangle_q W_{j,k} \langle h_k \rangle_q - \\
& - \sum_{j=d}^{D} \log \delta_j - \sum_{j=d}^{D} \frac{1}{2\delta_j^2} \left\langle x_j^2 \right\rangle_q + \sum_{j=d}^{D} \frac{\mu_j}{\delta_j^2} \langle x_j \rangle_q - \sum_{j=d}^{D} \frac{\mu_j^2}{2\delta_j^2} + \\
& + \sum_{k=1}^{H} \langle h_k \rangle_q \log \tau_k + \sum_{k=1}^{H} \langle 1 - h_k \rangle_q \log(1 - \tau_k) && \text{(A.6)} \\
=& K'(\boldsymbol{x}_{<d}) + \sum_{j=d}^{D} \frac{\mu_j^2 + \delta_j^2}{2\sigma_j^2} - \sum_{j=d}^{D} \frac{b_j \mu_j}{\sigma_j^2} - \sum_{k=1}^{H} c_k \tau_k - \\
& - \sum_{j=1}^{d-1} \sum_{k=1}^{H} \frac{1}{\sigma_j} \boldsymbol{x}_j W_{j,k} \tau_j - \sum_{j=d}^{D} \sum_{k=1}^{H} \frac{1}{\sigma_j} \mu_j W_{j,k} \tau_j - \\
& - \sum_{j=d}^{D} \log \delta_j + \sum_{k=1}^{H} \tau_k \log \tau_k + \sum_{k=1}^{H} (1 - \tau_k) \log(1 - \tau_k) && \text{(A.7)}
\end{aligned}$$

Taking partial derivatives and solving for zero:

$$\begin{aligned}
\frac{\partial \mathrm{KL}\left(q \parallel p\right)}{\partial \mu_i} =& 0 && \text{(A.8)} \\
\frac{\mu_i}{\delta_i^2} - \frac{b_i}{\delta_i^2} - \frac{1}{\delta_i} \sum_{k=1}^{H} W_{i,k} \tau_k =& 0 && \text{(A.9)} \\
\mu_i =& b_i + \delta_i \sum_{k=1}^{H} W_{i,k} \tau_k && \text{(A.10)}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathrm{KL}\left(q \parallel p\right)}{\partial \delta_i} =& 0 && \text{(A.11)} \\
\frac{\delta_i}{\sigma_i^2} - \frac{1}{\delta_i} =& 0 && \text{(A.12)} \\
\delta_i =& \sigma_i && \text{(A.13)}
\end{aligned}$$

$$\frac{\partial \mathrm{KL}\left(q \parallel p\right)}{\partial \tau_i} = 0 \tag{A.14}$$

$$\log \frac{\tau_i}{1-\tau_i} = c_i + \sum_{j=1}^{d-1} \frac{1}{\sigma_j} \boldsymbol{x}_j W_{j,i} + \sum_{j=d}^{D} \frac{1}{\sigma_j} \mu_j W_{j,i} \tag{A.15}$$

$$\tau_i = \mathrm{sigm}\left(c_i + \sum_{j=1}^{d-1} \frac{1}{\sigma_j} \boldsymbol{x}_j W_{j,i} + \sum_{j=d}^{D} \frac{1}{\sigma_j} \mu_j W_{j,i} \right) \tag{A.16}$$

Appendix B

RNADE-MoG gradient derivations

In this appendix we provide pseudo-code for the calculation of densities and learning gradients. No new material is presented.

B.1 Density estimation

In Algorithm 3 we detail the pseudocode for calculating the density of a datapoint under an RNADE with mixture of Gaussian conditionals. The model has parameters: $\boldsymbol{\rho} \in \mathbb{R}^D$, $\boldsymbol{W} \in \mathbb{R}^{D \times H}$, $\boldsymbol{c} \in \mathbb{R}^H$, $\boldsymbol{b}^\alpha \in \mathbb{R}^{D \times K}$, $\boldsymbol{V}^\alpha \in \mathbb{R}^{D \times H \times K}$, $\boldsymbol{b}^\mu \in \mathbb{R}^{D \times K}$, $\boldsymbol{V}^\mu \in \mathbb{R}^{D \times H \times K}$, $\boldsymbol{b}^\sigma \in \mathbb{R}^{D \times K}$, $\boldsymbol{V}^\sigma \in \mathbb{R}^{D \times H \times K}$

B.2 Learning gradients

Training of an RNADE model can be done using a gradient ascent algorithm on the log-likelihood of the model given the training data. Gradients can be calculated using automatic differentiation libraries (e.g. Theano Bergstra et al. (2010)). However we found our manual implementation to work faster in practice, possibly due to our recomputation of the $\boldsymbol{a}$ terms in the second *for* loop in Algorithm 4, which is more cache-friendly than storing them during the first loop.

Here we show the derivation of the gradients of each parameter of a NADE model with MoG conditionals. Following Bishop (1994), we define $\phi_i(x_d \,|\, \boldsymbol{x}_{<d})$ as the density of x_d under the i-th component of the conditional:

$$\phi_i(x_d \,|\, \boldsymbol{x}_{<d}) = \frac{1}{\sqrt{2\pi}\boldsymbol{\sigma}_{d,i}} \exp\left\{ -\frac{(x_d - \boldsymbol{\mu}_{d,i})^2}{2\boldsymbol{\sigma}_{d,i}^2} \right\}, \tag{B.1}$$

Algorithm 3 Computation of $p(\boldsymbol{x})$

$\boldsymbol{a} \leftarrow \boldsymbol{c}$
$p(\boldsymbol{x}) \leftarrow 1$
for d from 1 to D **do**
  $\boldsymbol{\psi}_d \leftarrow \rho_d \boldsymbol{a}$ ▷ Rescaling factors
  $\boldsymbol{h}_d \leftarrow \boldsymbol{\psi}_d \, \mathbf{1}_{\psi_d > 0}$ ▷ Rectified linear units
  $\boldsymbol{z}_{\boldsymbol{d}}^{\boldsymbol{\alpha}} \leftarrow \boldsymbol{V}_d^{\alpha\top} \boldsymbol{h}_d + \boldsymbol{b}_d^{\alpha}$
  $\boldsymbol{z}_{\boldsymbol{d}}^{\boldsymbol{\mu}} \leftarrow \boldsymbol{V}_d^{\mu\top} \boldsymbol{h}_d + \boldsymbol{b}_d^{\mu}$
  $\boldsymbol{z}_{\boldsymbol{d}}^{\boldsymbol{\sigma}} \leftarrow \boldsymbol{V}_d^{\sigma\top} \boldsymbol{h}_d + \boldsymbol{b}_d^{\sigma}$
  $\boldsymbol{\alpha}_d \leftarrow \operatorname{softmax}(\boldsymbol{z}_d^{\alpha})$ ▷ Enforce constraints
  $\boldsymbol{\mu}_d \leftarrow \boldsymbol{z}_d^{\mu}$
  $\boldsymbol{\sigma}_d \leftarrow \exp\{\boldsymbol{z}_d^{\sigma}\}$
  $p(\boldsymbol{x}) \leftarrow p(\boldsymbol{x}) p_{MoG}(x_d; \boldsymbol{\alpha}_d, \boldsymbol{\mu}_d, \boldsymbol{\sigma}_d)$
    ▷ p_{MoG} is the density of a mixture of Gaussians
  $\boldsymbol{a} \leftarrow \boldsymbol{a} + x_d \boldsymbol{W}_{d,\cdot}$ ▷ Activations are calculated recursively, x_d is a scalar
end for
return $p(\boldsymbol{x})$

and $\pi_i(x_d \,|\, \boldsymbol{x}_{<d})$ as the "responsibility" of the i-th component for x_d:

$$\pi_i(x_d \,|\, \boldsymbol{x}_{<d}) = \frac{\boldsymbol{\alpha}_{d,i} \phi_i(x_d \,|\, \boldsymbol{x}_{<d})}{\sum_{j=1}^{K} \boldsymbol{\alpha}_{d,j} \phi_j(x_d \,|\, \boldsymbol{x}_{<d})}. \tag{B.2}$$

It is easy to find just by taking their derivatives that:

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^{\alpha}} = \pi_i(x_d \,|\, \boldsymbol{x}_{<d}) - \boldsymbol{\alpha}_{d,i} \tag{B.3}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^{\mu}} = \pi_i(x_d \,|\, \boldsymbol{x}_{<d}) \frac{x_d - \boldsymbol{\mu}_{d,i}}{\boldsymbol{\sigma}_{d,i}^2} \tag{B.4}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^{\sigma}} = \pi_i(x_d \,|\, \boldsymbol{x}_{<d}) \left(\frac{(x_d - \boldsymbol{\mu}_{d,i})^2}{\boldsymbol{\sigma}_{d,i}^2} - 1 \right) \tag{B.5}$$

Using the chain rule we can calculate the derivative of the parameters of the

output layer parameters:

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{V}_d^\alpha} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\alpha} \frac{\partial \boldsymbol{z}_{d,i}^\alpha}{\boldsymbol{V}_d^\alpha} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\alpha} \boldsymbol{h} \tag{B.6}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{b}_d^\alpha} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\alpha} \frac{\partial \boldsymbol{z}_{d,i}^\alpha}{\boldsymbol{b}_d^\alpha} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\alpha} \tag{B.7}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{V}_d^\mu} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\mu} \frac{\partial \boldsymbol{z}_{d,i}^\alpha}{\boldsymbol{V}_d^\mu} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\mu} \boldsymbol{h} \tag{B.8}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{b}_d^\mu} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\mu} \frac{\partial \boldsymbol{z}_{d,i}^\alpha}{\boldsymbol{b}_d^\mu} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\mu} \tag{B.9}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{V}_d^\sigma} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\sigma} \frac{\partial \boldsymbol{z}_{d,i}^\alpha}{\boldsymbol{V}_d^\sigma} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\sigma} \boldsymbol{h} \tag{B.10}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{b}_d^\sigma} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\sigma} \frac{\partial \boldsymbol{z}_{d,i}^\alpha}{\boldsymbol{b}_d^\sigma} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\sigma} \tag{B.11}$$

By "backpropagating" the we can calculate the partial derivatives with respect to the output of the hidden units:

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{h}_d} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\alpha} \frac{\partial \boldsymbol{z}_{d,i}^\alpha}{\partial \boldsymbol{h}_d} + \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\mu} \frac{\partial \boldsymbol{z}_{d,i}^\mu}{\partial \boldsymbol{h}_d} + \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\sigma} \frac{\partial \boldsymbol{z}_{d,i}^\sigma}{\partial \boldsymbol{h}_d} \tag{B.12}$$

$$= \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\alpha} \boldsymbol{V}_d^\alpha + \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\mu} \boldsymbol{V}_d^\mu + \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{z}_{d,i}^\sigma} \boldsymbol{V}_d^\sigma \tag{B.13}$$

and calculate the partial derivatives with respect to all other parameters in RNADE:

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{\psi}_d} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{h}_d} \mathbf{1}_{\psi_d > 0} \tag{B.14}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \rho_d} = \sum_j \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{\psi}_{d,j}} \boldsymbol{a}_{d,j} \tag{B.15}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{a}_d} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{a}_{d+1}} + \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{h}_d} \rho_d \mathbf{1}_{\psi_d > 0} \tag{B.16}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{W}_{d,\cdot}} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{a}_d} x_d \tag{B.17}$$

$$\frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{c}} = \frac{\partial p(\boldsymbol{x})}{\partial \boldsymbol{a}_1} \tag{B.18}$$

Note that gradients are calculated recursively, due to (B.16), starting at $d = D$ and progressing down to $d = 1$.

Algorithm 4 Computation of the learning gradients for a datapoint $\boldsymbol{x}$

$\boldsymbol{a} \leftarrow \boldsymbol{c}$

for d from 1 to D **do** ▷ Compute the activation of the last dimension

$\boldsymbol{a} \leftarrow \boldsymbol{a} + x_d \boldsymbol{W}_{d,\cdot}$

end for

for d from D to 1 **do** ▷ Backpropagate errors

$\boldsymbol{\psi} \leftarrow \rho_d \boldsymbol{a}$ ▷ Rescaling factors

$\boldsymbol{h} \leftarrow \boldsymbol{\psi}\, \mathbf{1}_{\boldsymbol{\psi}>0}$ ▷ Rectified linear units

$\boldsymbol{z}^{\boldsymbol{\alpha}} \leftarrow \boldsymbol{V}_d^{\alpha\top} \boldsymbol{h} + \boldsymbol{b}_d^{\alpha}$

$\boldsymbol{z}^{\boldsymbol{\mu}} \leftarrow \boldsymbol{V}_d^{\mu\top} \boldsymbol{h} + \boldsymbol{b}_d^{\mu}$

$\boldsymbol{z}^{\boldsymbol{\sigma}} \leftarrow \boldsymbol{V}_d^{\sigma\top} \boldsymbol{h}_d + \boldsymbol{b}_d^{\sigma}$

$\boldsymbol{\alpha} \leftarrow \mathrm{softmax}(\boldsymbol{z}^{\alpha})$ ▷ Enforce constraints

$\boldsymbol{\mu} \leftarrow \boldsymbol{z}^{\mu}$

$\boldsymbol{\sigma} \leftarrow \exp\{\boldsymbol{z}^{\sigma}\}$

$\boldsymbol{\phi} \leftarrow \frac{1}{2}\frac{(\boldsymbol{\mu}-\boldsymbol{x}_d)^2}{\boldsymbol{\sigma}^2} - \log\boldsymbol{\sigma} - \frac{1}{2}\log(2\pi)$ ▷ Calculate gradients

$\boldsymbol{\pi} \leftarrow \frac{\boldsymbol{\alpha}\phi}{\sum_{j=1}^{K} \boldsymbol{\alpha}_j \phi_j}$

$\partial z^{\alpha} \leftarrow \boldsymbol{\pi} - \boldsymbol{\alpha}$

$\partial \boldsymbol{V}_d^{\alpha} \leftarrow \partial z^{\alpha} \boldsymbol{h}$

$\partial \boldsymbol{b}_d^{\alpha} \leftarrow \partial z^{\alpha}$

$\partial z^{\mu} \leftarrow \boldsymbol{\pi}(x_d - \boldsymbol{\mu})/\boldsymbol{\sigma}^2$

$\partial z^{\mu} \leftarrow \partial z^{\mu} * \sigma$ ▷ Move tighter components slower, allows higher learning rates

$\partial \boldsymbol{V}_d^{\mu} \leftarrow \partial z^{\mu} \boldsymbol{h}$

$\partial \boldsymbol{b}_d^{\mu} \leftarrow \partial z^{\mu}$

$\partial z^{\sigma} \leftarrow \boldsymbol{\pi}\{(x_d - \boldsymbol{\mu})^2/\boldsymbol{\sigma}^2 - 1\}$

$\partial \boldsymbol{V}_d^{\sigma} \leftarrow \partial z^{\sigma} \boldsymbol{h}$

$\partial \boldsymbol{b}_d^{\sigma} \leftarrow \partial z^{\sigma}$

$\partial \boldsymbol{h} \leftarrow \partial \boldsymbol{z}^{\alpha} \boldsymbol{V}_d^{\alpha} + \partial \boldsymbol{z}^{\mu} \boldsymbol{V}_d^{\mu} + \partial \boldsymbol{z}^{\sigma} \boldsymbol{V}_d^{\sigma}$

$\partial \boldsymbol{\psi} \leftarrow \partial \boldsymbol{h} \mathbf{1}_{\boldsymbol{\psi}>0}$ ▷ Second factor: indicator function with condition $\boldsymbol{\psi} > 0$

$\partial \rho_d \leftarrow \sum_j \partial \boldsymbol{\psi}_j a_j$

$\partial \boldsymbol{a} \leftarrow \partial \boldsymbol{a} + \partial \boldsymbol{\psi} \rho$

$\partial \boldsymbol{W}_{d,\cdot} \leftarrow \partial \boldsymbol{a} x_d$

if $d = 1$ **then**

$\partial \boldsymbol{c} \leftarrow \partial \boldsymbol{a}$

else

$\boldsymbol{a} \leftarrow \boldsymbol{a} - x_d \boldsymbol{W}_{d,\cdot}$

end if

end for

return $\partial\boldsymbol{\rho}$, $\partial\boldsymbol{W}$, $\partial\boldsymbol{c}$, $\partial\boldsymbol{b}^{\alpha}$, $\partial\boldsymbol{V}^{\alpha}$, $\partial\boldsymbol{b}^{\mu}$, $\partial\boldsymbol{V}^{\mu}$, $\partial\boldsymbol{b}^{\sigma}$, $\partial\boldsymbol{V}^{\sigma}$

Appendix C

Input masks and marginalization under a Bayes' classifier

In this appendix we show that interpreting each hidden unit of a neural network as a Bayes' classifier implies the need for a correction of their bias when input features are missing. This capability can be added to NADE by the use of input masks, as we proposed in Chapter 4.

A Bayes' classifier calculates the posterior probability of a datapoint belonging to a particular class by comparing its log-density under a generative model of each class (e.g. Duda et al., 2001). Bayes' classifiers can always be expressed as a sigmoid function:

$$p(A \,|\, \boldsymbol{x}) = \frac{p_A(\boldsymbol{x})\pi_A}{p_A(\boldsymbol{x})\pi_A + p_B(\boldsymbol{x})\pi_B} \tag{C.1}$$

$$= \text{sigm}\left(\log p_A(\boldsymbol{x}) - \log p_B(\boldsymbol{x}) + \log \frac{\pi_A}{\pi_B}\right) \tag{C.2}$$

Where $p_A(\boldsymbol{x})$ and $p_B(\boldsymbol{x})$ are the density functions of classes A and B, and π_A and π_B the prior probabilities of each class.

Thus, each hidden unit in a neural network with sigmoid non-linearities can be interpreted as a Bayes' classifier where $\log p_A(\boldsymbol{x}) - \log p_B(\boldsymbol{x})$ is a linear function. This is the case, for example, when the generative models of each class are multivariate Bernoulli distributions, in the case of binary data, or multivariate Gaussians with shared covariance in the case of real-valued data.

Assuming $\boldsymbol{x}$ is a multivariate binary variable, and classes A and B are modelled by multivariate Bernoulli distributions, let us denote by $\mu_d^{(A)}$ the probability of the d-th dimension taking value 1 under the generative model for class A, and similarly $\mu_d^{(B)}$ for class B. The probability of an observation under model A is

given by:

$$\log p(\boldsymbol{x} \mid A) = \sum_d \left[x_d \log \mu_d^{(A)} + (1 - x_d) \log \left(1 - \mu_d^{(A)} \right) \right] \tag{C.3}$$

$$= \sum_d \left[x_d \log \frac{\mu_d^{(A)}}{1 - \mu_d^{(A)}} + \log \left(1 - \mu_d^{(A)} \right) \right], \tag{C.4}$$

and similarly for B. Therefore, under multivariate Bernoulli models, (C.2) can be written as:

$$p(A \mid \boldsymbol{x}) = \operatorname{sigm} \left(\sum_d x_d \underbrace{\left[\log \frac{\mu_d^{(A)}}{1 - \mu_d^{(A)}} - \log \frac{\mu_d^{(B)}}{1 - \mu_d^{(B)}} \right]}_{w_d} + \underbrace{\sum_d \log \frac{1 - \mu_d^{(A)}}{1 - \mu_d^{(B)}} + \log \frac{\pi_A}{\pi_B}}_{b} \right) \tag{C.5}$$

Where the factors multiplying x_d correspond to the *weights*, w_d, of a hidden unit and the second sum inside the sigmoid corresponds to the *bias*, b.

Given the factorial nature of the multivariate-Bernoulli, marginalizing any missing variables, simply requires ignoring the terms corresponding to those dimensions. In (C.5), this means that the sums over d must iterate only across the dimensions present. Each dimension present adds a $\log \frac{1-\mu_d^{(A)}}{1-\mu_d^{(B)}}$ to the bias. Therefore, in order to marginalize the missing dimensions, a correction of the bias in the discriminant function is required.

The inclusion of input masks in the orderless-NADE adds exactly this capability to the model. We can interpret the *mask weights*, U in (4.7) as $\log \frac{1-\mu_d^{(A)}}{1-\mu_d^{(B)}}$. Therefore, after the addition of input masks, each of the hidden units in the orderless-NADE can be interpreted as a Bayes' classifier that performs correct inference even with missing inputs.

The parameters of each hidden unit in an orderless-NADE can, therefore, be parameterized in terms of the expected value for each multivariate Bernoulli and the log prior-odds as in (C.5), or in terms of W, U and b, as in (4.7) on page 71.

In the case of real-valued inputs, similar derivations can be reached in the case of a Bayes' classifier that utilizes a shared diagonal covariance Gaussian as a generative model of each class. That is,

$$\log p(\boldsymbol{x} \mid A) = \frac{1}{2} (\boldsymbol{x} - \mu_A)^\top \Sigma^{-1} (\boldsymbol{x} - \mu_A) - \log Z \tag{C.6}$$

In which case, given the shared covariance, the normalizing and quadratic terms in the discriminant cancel and we end up with a linear discriminant function:

$$p(A \,|\, x) = \text{sigm}\left(\boldsymbol{x}^\top \Sigma^{-1}(\mu_A - \mu_B) + \frac{1}{2}\left[\mu_B^\top \Sigma^{-1}\mu_B - \mu_A^\top \Sigma^{-1}\mu_A\right] + \log\frac{\pi_A}{\pi_B}\right) \quad \text{(C.7)}$$

Which in the case of diagonal covariance can be rewritten as a sum of simple terms:

$$p(A \,|\, x) = \text{sigm}\left(\sum_d x_d \underbrace{\frac{\mu_d^{(A)} - \mu_d^{(B)}}{\sigma_d^2}}_{w_d} + \underbrace{\sum_d \frac{1}{2}\left[\left(\frac{\mu_d^{(B)}}{\sigma_d}\right)^2 - \left(\frac{\mu_d^{(A)}}{\sigma_d}\right)^2\right] + \log\frac{\pi_A}{\pi_B}}_{b}\right) \quad \text{(C.8)}$$

For the shared diagonal covariance case, marginalizing out missing dimensions is equivalent to iterating the sums over d across the inputs present. As in the binary case, each input present adds a term to the bias of the sigmoid-linear discriminant function, and a bias correction is necessary when some of the dimensions are missing. Orderless-RNADE with input masks has the ability to correctly marginalize missing inputs. We note that in this case it is not possible to recover the parameters of the generative model for each class from the parameters of the orderless-RNADE.

This last derivation points to a route for a more powerful first hidden layer, by using non-shared or non-diagonal covariance matrices.

In all our experiments we have used rectified linear units instead of sigmoid units. However, we can think of the ReLU non-linearity as an approximation to $-\log\text{sigm}(-x)$ (see Figure C.1 on the following page), and the output of ReLU units as the negative log-posterior of the a two-class classifier.

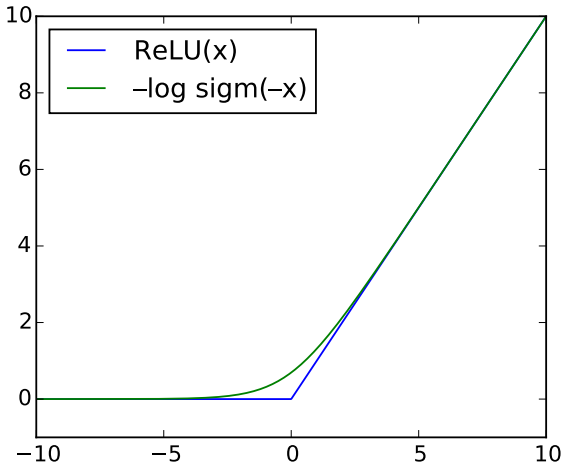

Figure C.1: Comparison of rectified linear (blue) and negative log sigmoid (green) non-linearities.

Bibliography

Ackley, D., Hinton, G., and Sejnowski, T. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169. (pages 9 and 18)

Bache, K. and Lichman, M. (2013). UCI machine learning repository. http://archive.ics.uci.edu/ml. (pages 38, 49, 76, and 85)

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press. (pages 8, 12, and 48)

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv preprint arXiv:1506.03099*. (page 108)

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127. (page 68)

Bengio, Y. (2011). Discussion of "The Neural Autoregressive Distribution Estimator". In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15 of *Journal of Machine Learning Research W&CP*, pages 38–39. (page 48)

Bengio, Y. and Bengio, S. (2000). Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems*, volume 12, pages 400–406. MIT Press. (pages 34, 42, and 73)

Bengio, Y., Laufer, E., Alain, G., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Journal of Machine Learning Research W&CP*, pages 226–234. (pages 30 and 31)

Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013). Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, volume 26, pages 899–907. MIT Press. (pages 30 and 31)

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305. (page 65)

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. (page 115)

Billingsley, P. (2008). *Probability and Measure*. John Wiley & Sons. (page 5)

Bishop, C. (1994). Mixture density networks. Technical Report NCRG 4288, Neural Computing Research Group, Aston University, Birmingham. (pages 45, 64, 94, and 115)

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Inc. (page 94)

Bishop, C. (2006). *Pattern Tecognition and Machine Learning*. Springer New York. (page 33)

Bishop, C., Svensén, M., and Williams, C. (1998). GTM: the generative topographic mapping. *Neural computation*, 10(1):215–234. (page 27)

Black, A. and Tokuda, K. (2005). The Blizzard Challenge 2005: Evaluating corpus-based speech synthesis on common datasets. In *Proceedings of the 6th Annual Conference of the International Speech Communication Association*, pages 77–80. ISCA. (page 92)

Bornschein, J. and Bengio, Y. (2014). Reweighted wake-sleep. *CoRR*. (pages 7, 29, and 108)

Bridle, J. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer. (page 94)

Buntine, W. (1991). Theory refinement on Bayesian networks. In *Proceedings of the 7th conference on Uncertainty in Artificial Intelligence*, pages 52–60. Morgan Kaufmann Publishers. (page 63)

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. volume abs/1509.00519. (page 78)

Casella, G. and Berger, R. (2002). *Statistical Inference*. Duxbury Press. (page 5)

Cho, K., Ilin, A., and Raiko, T. (2011). Improved learning of Gaussian-Bernoulli restricted Boltzmann machines. In *Proceedings of the 21st International Conference on Artificial Neural Networks*, pages 10–17. Springer. (pages 22 and 44)

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467. (page 63)

Cooke, M., Mayo, C., Valentini-Botinhao, C., Stylianou, Y., Sauert, B., and Tang, Y. (2013). Evaluating the intelligibility benefit of speech modifications in known noise conditions. *Speech Communication*, 55:572–585. (page 101)

Courville, A., Bergstra, J., and Bengio, Y. (2011). A spike and slab restricted Boltzmann machine. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15 of *Journal of Machine Learning Research W&CP*, pages 233–241. (page 22)

Dahl, G., Sainath, T., and Hinton, G. (2013). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8609–8613. IEEE. (page 21)

Dayan, P. and Hinton, G. (1996). Varieties of Helmholtz machine. *Neural Networks*, 9(8):1385–1403. (pages 27 and 29)

Dayan, P., Hinton, G., Neal, R., and Zemel, R. (1995). The Helmholtz machine. *Neural Computation*, 7(5):889–904. (page 27)

de Finetti, B. (1974). *Theory of Probability*, volume 1. Wiley. (page 1)

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B: Statistical Methodology*, 39(1):1–38. (pages 11 and 12)

Duda, R., Hart, P., and Stork, D. (2001). *Pattern Classification*. John Wiley & Sons. (pages 6, 51, and 119)

Fisher, R. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, pages 309–368. (pages 2 and 7)

Frey, B. (1998). *Graphical Models for Machine Learning and Digital Communication*. MIT Press. (pages 8, 31, 33, and 73)

Frey, B., Hinton, G., and Dayan, P. (1996). Does the wake-sleep algorithm produce good density estimators? In *Advances in Neural Information Processing Systems*, volume 8, pages 661–670. MIT Press. (page 30)

Gales, M. (1999). Semi-tied covariance matrices for hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 7(3):272–281. (page 94)

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., and Zue, V. (1993). Darpa TIMIT acoustic-phonetic continuous speech corpus CD-ROM. (page 60)

Gelman, A. and Shalizi, C. (2010). Philosophy and the practice of Bayesian statistics. *British Journal of Mathematical and Statistical Psychology*. (page 2)

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741. (page 20)

Ghahramani, Z. and Hinton, G. (1996). The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, University of Toronto. (pages 16, 18, and 51)

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Journal of Machine Learning Research W&CP*, pages 1319–1327. (pages 73 and 109)

Gordon, D. and Desjardins, M. (1995). Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1-2):5–22. (page 2)

Goroshin, R. and LeCun, Y. (2013). Saturating auto-encoders. *CoRR*. arXiv preprint arXiv:1301.3577. (page 109)

Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep autoregressive networks. In *Proceedings of The 31st International Conference on Machine Learning*, volume 32 of *Journal of Machine Learning Research W&CP*, pages 1242–1250. (page 30)

Gregor, K. and LeCun, Y. (2011). Learning representations by maximizing compression. *CoRR*. arXiv preprint arXiv:1108.1169. (pages 36 and 73)

Grosse, R., Maddison, C., and Salakhutdinov, R. (2013). Annealing between distributions by averaging moments. In *Advances in Neural Information Processing Systems*, volume 26, pages 2769–2777. (page 20)

Guyon, I., Aliferis, C., and Elisseeff, A. (2007). Causal feature selection. *Computational methods of feature selection*, pages 63–86. (page 63)

Henter, G., Merritt, T., Shannon, M., Mayo, C., and King, S. (2014). Measuring the perceptual effects of modelling assumptions in speech synthesis using stimuli constructed from repeated natural speech. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association, Interspeech*, pages 1504–1508. (page 93)

Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800. (page 21)

Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 201003, Department of Computer Science, University of Toronto. (page 20)

Hinton, G., Dayan, P., Frey, B., and Neal, R. (1995). The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161. (pages 27 and 28)

Hinton, G., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554. (pages 23, 24, and 25)

Hinton, G. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1:282–317. (pages 9 and 18)

Hsu, D. and Kakade, S. (2013). Learning mixtures of spherical Gaussians: moment methods and spectral decompositions. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 11–20. ACM. (page 11)

Hunt, A. and Black, A. (1996). Unit selection in a concatenative speech synthesis system using a large speech database. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-96*, volume 1, pages 373–376. IEEE. (page 92)

ITU (2003). Method for the subjective assessment of intermediate quality level of coding systems. Technical Report BS.1534, International Telecommunication Union Radiocommunication Assembly. (page 94)

Jaworski, P., Durante, F., Hardle, W., and Rychlik, T. (2010). *Copula Theory and its Applications*. Springer. (page 8)

Jaynes, E. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. (page 1)

Jeffreys, H. (1939). *Theory of Probability*. Clarendon Press. (page 1)

Jimenez-Rezende, D., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of The 31st International Conference on Machine Learning*, volume 32 of *Journal of Machine Learning Research W&CP*, pages 1278–1286. (page 30)

Jones, M. C. and Pewsey, A. (2009). Sinh-arcsinh distributions. *Biometrika*, 96(4):761–780. (page 48)

Jordan, M., Ghahramani, Z., and Saul, L. (1997). Hidden Markov Decision Trees. In *Advances in Neural Information Processing Systems*, volume 9. MIT Press. (page 95)

Kawahara, H., Masuda-Katsuse, I., and de Cheveigné, A. (1999). Restructuring speech representations using a pitch-adaptive time–frequency smoothing and an instantaneous-frequency-based f0 extraction: Possible role of a repetitive structure in sounds. *Speech Communication*, 27(3):187–207. (page 101)

Kingma, D. and Welling, M. (2014). Auto-encoding variational bayes. *CoRR*. arXiv preprint arXiv:1312.6114. (page 30)

Koenker, R. and Bassett, G. (1978). Regression quantiles. *Econometrica: journal of the Econometric Society*, pages 33–50. (page 108)

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. (pages 7 and 8)

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto, Canada. (page 22)

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. (page 2)

Larochelle, H. and Lauly, S. (2012). A neural autoregressive topic model. In *Advances in Neural Information Processing Systems*, volume 25, pages 2708–2716. (page 39)

Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15, pages 29–37. (pages 3, 20, 21, 36, 37, 38, 49, 67, 73, 75, 76, 77, and 78)

LeCun, Y., Bottou, L., Orr, G., and Müller, K. (1998). Efficient backprop. *Neural networks: Tricks of the trade*, pages 546–546. (page 38)

LeCun, Y., Chopra, S., Hadsell, R., Ranzatto, M., and Huang, F. (2007). Energy-based models. In *Predicting Structured Data*, pages 191–246. MIT Press. (page 8)

MacKay, D. (1995a). Bayesian methods for supervised neural networks. In *The Handbook of Brain Theory and Neural Networks*, pages 144–149. MIT Press. (page 7)

MacKay, D. (1995b). Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 354(1):73–80. (pages 26 and 29)

MacKay, D. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. (pages 2, 7, 12, and 89)

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1):77–91. (pages 1 and 41)

Martens, J. (2010). Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742. (pages 21 and 109)

Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *International Conference on Computer Vision*, volume 2, pages 416–423. IEEE. (page 52)

Mitchell, T. (1980). The need for biases in learning generalizations. Technical report, Department of Computer Science, Laboratory for Computer Science Research, Rutgers University. (page 2)

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. In *Proceedings of the 31th International Conference on Machine Learning*, volume 32 of *Journal of Machine Learning Research W&CP*, pages 1791–1799. (pages 29 and 30)

Mohamed, A., Dahl, G., and Hinton, G. (2009). Deep belief networks for phone recognition. In *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*. (page 2)

Murphy, K. (2012). *Machine learning: a probabilistic perspective*. MIT press. (page 9)

Murray, I. (2015). Personal communication. (page 78)

Murray, I. and Salakhutdinov, R. (2009). Evaluating probabilities under high-dimensional latent variable models. In *Advances in Neural Information Processing Systems*, volume 21, pages 1137–1144. (pages 20, 25, 26, and 78)

Nair, V. and Hinton, G. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814. (pages 21, 49, 76, 101, and 109)

Neal, R. (1995). *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto. (page 7)

Ormoneit, D. and Tresp, V. (1995). Improved Gaussian mixture density estimates using Bayesian penalty terms and network averaging. In *Advances in Neural Information Processing Systems*, volume 8, pages 542–548. MIT Press. (page 72)

Ozair, S., Yao, L., and Bengio, Y. (2013). Multimodal transitions for generative stochastic networks. *CoRR*. arXiv preprint arXiv:1312.5578. (page 108)

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann. (page 27)

Qian, Y., Fan, Y., Hu, W., and Soong, F. (2014). On the training aspects of deep neural network (DNN) for parametric TTS synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP-2014*, pages 3829–3833. (pages 93 and 101)

Raiko, T., Li, Y., Cho, K., and Bengio, Y. (2014). Iterative neural autoregressive distribution estimator NADE-k. In *Advances in Neural Information Processing Systems*, volume 27, pages 325–333. (page 108)

Ranzato, M. and Hinton, G. (2010). Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *Computer Vision and Pattern Recognition*, pages 2551–2558. IEEE. (page 22)

Richmond, K., King, S., and Taylor, P. (2003). Modelling the uncertainty in recovering articulation from acoustics. *Computer Speech & Language*, 17(2):153–172. (page 45)

Robinson, T. (1994). SHORTEN: simple lossless and near-lossless waveform compression. Technical Report CUED/F-INFENG/TR.156, Engineering Department, Cambridge University. (page 48)

Rubin, D. and Thayer, D. (1982). EM algorithms for ML factor analysis. *Psychometrika*, 47(1):69–76. (page 17)

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455. MIT Press. (pages 25 and 26)

Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of deep belief networks. In *Proceedings of the 25th International Conference on Machine Learning*, pages 872–879. (pages 20, 21, 49, and 78)

Schaul, T., Zhang, S., and LeCun, Y. (2013). No more pesky learning rates. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Journal of Machine Learning Research W&CP*, pages 343–351. (pages 65 and 109)

Senior, A., Heigold, G., Ranzato, M., and Yang, K. (2013). An empirical study of learning rates in deep neural networks for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP-2013*, pages 6724–6728. (page 102)

Shannon, M., Zen, H., and Byrne, W. (2011). The effect of using normalized models in statistical speech synthesis. In *Proceedings of the 12th Annual Conference of the International Speech Communication Association, Interspeech*, pages 121–124. (pages 93, 94, and 100)

Shannon, M., Zen, H., and Byrne, W. (2013). Autoregressive models for statistical parametric speech synthesis. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(3):587–597. (pages 100 and 105)

Silva, R., Blundell, C., and Teh, Y. (2011). Mixed cumulative distribution networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15, pages 670–678. (page 49)

Silverman, B. (1986). *Density Estimation for Statistics and Data Analysis*, volume 26. CRC press. (pages 5, 7, and 8)

Smyth, P. and Wolpert, D. (1999). Linearly combining density estimators via stacking. *Machine Learning*, 36(1-2):59–83. (pages 72 and 89)

Snoek, J., Larochelle, H., and Adams, R. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, pages 2960–2968. (pages 65 and 109)

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, and Adams, R. (2015). Scalable Bayesian optimization using deep neural networks. *CoRR*. arXiv preprint arXiv:1502.05700. (page 109)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958. (pages 65, 74, and 109)

Sutskever, I. (2013). *Training Recurrent Neural Networks*. PhD thesis, University of Toronto. (pages 21 and 76)

Tang, Y., Salakhutdinov, R., and Hinton, G. (2012). Deep mixtures of factor analysers. In *Proceedings of the 29th International Conference on Machine Learning*, pages 505–512. (pages 24, 49, and 52)

Taskar, B. (2004). *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University. (page 1)

Theis, L., Gerwinn, S., Sinz, F., and Bethge, M. (2011). In all likelihood, deep belief is not enough. *The Journal of Machine Learning Research*, 12:3071–3096. (page 42)

Theis, L., Hosseini, R., and Bethge, M. (2012). Mixtures of conditional Gaussian scale mixtures applied to multiscale image representations. *PLoS ONE*, 7(7). (page 48)

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. *Proceedings of the 25th International Conference on Machine Learning*, pages 1064–1071. (page 21)

Tokuda, K., Yoshimura, T., Masuko, T., Kobayashi, T., and Kitamura, T. (2000). Speech parameter generation algorithms for HMM-based speech synthesis. In *IEEE International Conference on Acoustics, Speech, and Signal Processing. ICASSP-2000*, pages 1315–1318. (pages 93 and 105)

Tomoki, T. and Tokuda, K. (2007). A speech parameter generation algorithm considering global variance for HMM-based speech synthesis. *IEICE Transactions on Information and Systems*, 90(5):816–824. (page 93)

Uria, B. (2011). A deep belief network for the acoustic-articulatory inversion mapping problem. Master's thesis, University of Edinburgh. (page 45)

Uria, B., Murray, I., and Larochelle, H. (2013). RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, volume 26, pages 2175–2183. (page 41)

Uria, B., Murray, I., and Larochelle, H. (2014). A deep and tractable density estimator. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Journal of Machine Learning Research W&CP*, pages 467–475. (page 67)

Uria, B., Murray, I., Renals, S., Valentini, C., and Bridle, J. (2015). Modelling acoustic feature dependencies with artificial neural networks: Trajectory-RNADE. In *IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP-2015*, pages 4465–4469. (page 91)

Verbeek, J. (2005). Mixture of factor analyzers Matlab implementation. (page 51)

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103. ACM. (pages 30 and 74)

Wang, N., Melchior, J., and Wiskott, L. (2012). An analysis of Gaussian-binary restricted Boltzmann machines for natural images. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 287–292. (page 23)

Welling, M. and Hinton, G. (2002). A new learning algorithm for mean-field Boltzmann machines. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 351–357. Springer. (page 36)

Welling, M., Rosen-Zvi, M., and Hinton, G. (2005). Exponential family harmoniums with an application to information retrieval. *Advances in Neural Information Processing Systems*, 17:1481–1488. (page 22)

Williams, C. (2005). How to pretend that correlated variables are independent by using difference observations. *Neural Computation*, 17(1):1–6. (page 100)

Williams, P. (1996). Using neural networks to model conditional multivariate densities. *Neural Computation*, 8(4):843–854. (pages 45 and 94)

Wolpert, D. (1996). The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390. (page 2)

Xie, F., Qian, Y., Fan, Y., Soong, F., and Li, H. (2014). Sequence error minimization training of neural network for voice conversion. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association, Interspeech*, pages 2283–2287. (page 100)

Yin, X., Ling, Z., and Dai, L. (2014). Spectral modeling using neural autoregressive distribution estimators for statistical parametric speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP-2014*, pages 3824–3828. (page 105)

Zen, H., Nose, T., Yamagishi, J., Sako, S., Masuko, T., Black, A., and Tokuda, K. (2007a). The HMM-based speech synthesis system version 2.0. In *Proceedings of the 6th ISCA Workshop on Speech Synthesis*, pages 294–299. (page 101)

Zen, H. and Senior, A. (2014). Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP-2014*, pages 3872–3876. IEEE. (pages 93, 94, and 101)

Zen, H., Senior, A., and Schuster, M. (2013). Statistical parametric speech synthesis using deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP-2013*, pages 7962–7966. IEEE. (page 93)

Zen, H., Tokuda, K., and Black, A. (2009). Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064. (pages 92 and 93)

Zen, H., Tokuda, K., and Kitamura, T. (2004). An introduction of trajectory model into HMM-based speech synthesis. In *Proceedings of the 5th ISCA Workshop on Speech Synthesis*, pages 191–196. (pages 93, 96, and 99)

Zen, H., Tokuda, K., and Kitamura, T. (2007b). Reformulating the HMM as a trajectory model by imposing explicit relationships between static and dynamic feature vector sequences. *Computer Speech & Language*, 21(1):153–173. (page 93)

Zoran, D. (2013). Personal communication. (page 52)

Zoran, D. and Weiss, Y. (2011). From learning models of natural image patches to whole image restoration. In *International Conference on Computer Vision*, pages 479–486. IEEE. (pages 7, 14, 52, and 55)

Zoran, D. and Weiss, Y. (2012). Natural images, Gaussian mixtures and dead leaves. *Advances in Neural Information Processing Systems*, 25:1745–1753. (pages 49, 52, 56, 85, and 88)